VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PEER-TO-PEER WEBOVÉ SDÍLENÍ SOUBORŮ VE SKUPINĚ

PEER-TO-PEER WEB FILE SHARING IN A GROUP

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. FILIP POKORNÝ**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání diplomové práce/21214/2017/xpokor62

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Pokorný Filip, Bc.**
Obor: Informační systémy
Téma: **Peer-To-Peer webové sdílení souborů ve skupině**
Peer-To-Peer Web File Sharing in a Group
Kategorie: Uživatelská rozhraní

Pokyny:

1. Seznamte se s problematikou tvorby webových aplikací; zaměřte se na jednoduché služby s maximálně jednoduchým a přístupným ovládáním.
2. Seznamte se s problematikou přenosu souborů bez použití serveru, tj. peer-to-peer. Vyhledejte a popište relevantní nástroje.
3. Vyhledejte a analyzujte existující aplikace řešící problém podobný tomu zadanému.
4. Navrhněte funkčnost webové aplikace, která umožní přenášet soubory mezi klienty ve skupině.
5. Prototypujte jednotlivé prvky uživatelského rozhraní i přenosu souborů a testujte je.
6. Navrhněte a implementujte webovou aplikaci pro přenos souborů ve skupině.
7. Testujte vytvořenou aplikaci a iterativně ji vylepšujte na základě výsledků testů.
8. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN-13: 978-0321965516
- Susan M. Weinschenk: 100 věcí, které by měl každý designér vědět o lidech, Computer Press, Brno 2012
- Jan Řezáč: Web ostrý jako břitva, Baroque Partners, 2014
- Alan B Johnston, Daniel C Burnett: WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web, Digital Codex LLC, 2014

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 5, značné rozpracování bodu 6.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Herout Adam, prof. Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

V rámci této práce jsem se zaměřil na řešení problému sdílení souborů mezi uživateli prostřednictvím peer-to-peer přenosu ve webovém prostředí. Cílem bylo vytvořit jednoduchou aplikaci, která umožní skupině uživatelů sdílet soubory mezi sebou bez účasti třetích stran a jakýchkoliv limitů. Tento cíl jsem realizoval prostřednictvím moderních webových technologií, z nichž nejdůležitější byla knihovna WebRTC, prostřednictvím které byl implementován peer-to-peer přenos. Tyto technologie byly podpořeny optimalizovanými algoritmy a nejaktuálnějšími prostředky pro zpracování a ukládání sdílených souborů. Výsledkem je real-time aplikace umožňující uživatelům vytvářet místnosti v nichž sdílí své soubory. Toto řešení bylo podrobeno rychlostním testům, kde v porovnání s řešeními podobného charakteru a jinými běžně používanými prostředky pro přenos souborů, vyšlo v testovaném prostředí jako nejrychlejší.

Abstract

The diploma thesis deals with file sharing among users through peer-to-peer transfer in a web environment. The aim was to create a simple application which would provide a group of users with an opportunity to share files among them without any presence of third party or limitation. I have met the target using modern web technologies, of which the most important was WebRTC library through which peer-to-peer transfer has been implemented. These technologies were supported by optimased algorithm and the most recent tools for processing and storage of shared files. The result is a real-time application enabling users to create rooms, in which they can share their files. This product was subjected to speed tests, thanks to which the application turned out to be the fastest in the tested environment contrary to solutions with similar nature or any other commonly used tools for file sharing.

Klíčová slova

webová aplikace, real-time synchronizace, peer-to-peer, sdílení souborů, ve skupině, Polymer, WebRTC, Parse, Socket.io, JavaScript, FileSystem API, StreamSaver

Keywords

web application, real-time synchronization, peer-to-peer, file sharing, in group, Polymer, WebRTC, Parse, Socket.io, JavaScript, FileSystem API, StreamSaver

Citace

POKORNÝ, Filip. *Peer-To-Peer webové sdílení souborů ve skupině*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Peer-To-Peer webové sdílení souborů ve skupině

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením prof. Ing. Adam Herout, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Filip Pokorný
22. května 2018

Poděkování

Na prvním místě bych rád poděkoval svému vedoucímu práce prof. Ing. Adamu Heroutovi, Ph.D., za jeho odborné vedení a připomínky. Dále bych rád poděkoval rodině a přítelkyni za podporu, díky které mi bylo umožněno práci dokončit.

Obsah

Kapitola 1

Úvod

V dnešní době lidé disponují velkým množstvím zařízení, která prostřednictvím svých vstupů zaznamenávají kvanta nových informací. Tato data je zapotřebí jednoduše, rychle a bezpečně sdílet napříč dalšími zařízeními.

Existuje vcelku spousta způsobů, jak svá data přenést, ale žádný z nich není ideální. Příkladem můžou být paměťová média jako flash paměti, externí disky atd., jež jsou ale omezena velikostí a dosahem. Při využití sítě, kterou je většina zařízení propojena, se nejčastěji lze setkat s uložením posílaných dat na úložiště třetí strany, odkud jsou data následně stažena. V tomto způsobu se ale objevuje mnoho zádrhelů, jako např. omezení ve velikosti posílaných souborů, rychlosti přenosu a v neposlední řadě i v samotné otázce bezpečnosti, neboť ve třetí stranu vkládáme svoji důvěru.

Z tohoto důvodu jsem se rozhodl vytvořit webovou aplikaci pro sdílení dat ve skupině prostřednictvím peer-to-peer přenosu. Nově vytvořená aplikace si za hlavní cíle a hodnoty klade rychlost, jednoduchost, bezpečnost a elegantnost. Aplikace je koncipována ke shlukování uživatelů do skupin, ve kterých si navzájem mohou sdílet svá data. Samotný přenos je navržen jako peer-to-peer, díky čemuž data proudí pouze mezi zdrojovým a cílovým zařízením nejkratší cestou v síti, čímž by mělo být docíleno velkých rychlostí. A závěrem její samotné umístění do webového prostředí umožní takovou službu použít skoro na jakémkoliv zařízení bez nutnosti instalace, a tím je dosaženo i snadného sdílení.

Za účelem vytvoření této aplikace byly zmapovány technologie a prostředky pro její vytvoření, čemuž se věnuje celá druhá kapitola. Vzhledem k umístění do webového prostředí jsou tyto technologie orientované pro použití v rámci jazyka JavaScript.

Třetí kapitola je věnována analýze a zhodnocení již existujících obdobných řešení. Zde se lze dočíst hlavní nedostatky, ale i zajímavé myšlenky či prvky zkoumaných řešení. Závěrem je shrnuto, co z analýzy pro nově vznikající koncept vyplývá. Následující kapitola navrhuje a specifikuje novou aplikaci, která je předmětem a cílem předkládané diplomové práce.

Po navržení aplikace následuje čtvrtá kapitola popisující samotnou implementaci. V první části kapitoly je detailně popsáno grafické zpracování a uživatelské rozhraní. Dále následuje část věnující se zpracování souborů a složek, čímž je myšlena příprava a odesílání souborů na straně odesílatele a následné zpracování datového toku na straně příjemce, které se ukázalo jako největší komplikace v rámci celé implementace. Následně je rozebrána serverová část celého konceptu, jež se stará o synchronizaci všech připojených relací, bezpečnost a ukládání dat. Závěrem se lze dočíst o dalším vývoji, který lze realizovat.

Na úspěšnou implementaci navazuje testování zaměřující se na dva hlavní předměty. Prvním předmětem testování bylo uživatelské rozhraní, které zprvu testovala vybraná skupina uživatelů, jejichž ovládání uživatelského rozhraní a následné připomínky byly podnětem

pro úpravu a přizpůsobení celé aplikace pro snadnější a intuitivnější ovládání. Následovalo zveřejnění s podporou zanechání zpětné vazby. Na základě obdržené zpětné vazby došlo k dalšímu ladění a přizpůsobení pro potřeby uživatelů. Druhým předmětem testování byla výkonnostní stránka, v rámci níž jsem porovnal obdobné aplikace a alternativní způsoby přenosu dat, v nichž implementovaná aplikace vyšla nejlépe.

Kapitola 2

Tvorba webové aplikace

V této kapitole se věnuji principům a technologiím, které napomáhají k tvorbě moderních webových aplikací. Dále zde rozebírám omezení, jimiž jsou webové aplikace v současné době limitovány. Obsah kapitoly je směřován k pozdějšímu návrhu a implementaci cílové aplikace pro peer-to-peer sdílení souborů ve skupině.

2.1 Aplikační rámce jazyka JavaScript

Aplikační rámce, což je překlad více obvyklého pojmu framework, jsou skoro nezbytností při tvorbě větších webových aplikací postavené na jazyce JavaScript [23]. Jedná se o šablonovací systémy, které vytváří kostru aplikace, definují strukturu a uspořádání modulů. Vstupem bývají strukturovaná data a pseudo-HTML šablona, jejichž následnou substitucí je vrácen výsledný HTML dokument. Programátor je tak veden k použití osvědčených návrhových a architektonických vzorů, například MVC a MVP.

V rámci nově vznikající aplikace větších rozměrů je zapotřebí přemýšlet o výběru mezi dostupnými aplikačními rámci. V nynější době lze vybírat z vcelku velkého množství o různých vlastnostech. K nejpopulárnějším patří AngularJS[1], React[2] nebo Vue[3] [3]. Osobně jsem zvolil aplikační rámec Polymer[4], který dále detailněji popisuji a rozebírám. Všechny zmíněné aplikační rámce sdílí možnost tvorby vlastních HTML prvků, což do určité míry dovoluje vývojáři použít deklarativní programování oproti imperativnímu. Setkáváme se s *data-bindingem* umožňujícím jednoduchou a efektivní distribuci změn do datového modelu. Další významnou výhodou při použití aplikačního rámce je bezpochyby napomáhání k překonání rozdílů mezi jednotlivými webovými prohlížeči.

2.1.1 Polymer

Jedná se o vcelku mladý JavaScriptový aplikační rámec, jenž byl založen v květnu roku 2015 týmem vývojářů pracujících v rámci organizace Chrome [7]. Ta spadá pod společnost Google, která se tak zároveň stará i o jeho propagaci. Polymer je široce využíván, příkladem může být největší server pro sdílení videosouborů YouTube[5], jehož front-end je vytvářen právě v něm [33].

[1] https://angular.io/
[2] https://reactjs.org/
[3] https://vuejs.org/
[4] https://www.polymer-project.org/
[5] https://www.youtube.com/

Hlavním cílem projektu je napomoct vývojářům vytvářet progresivní webové aplikace, čemuž napomáhá podpora platforem Web Components[6], Service Workers a HTTP/2 [1]. Web Components je souhrnné označení pro standardy W3C[7], jejichž hlavními zástupci jsou Custom Element (definice vlastních HTML prvků), Shadow DOM (zapouzdření DOM a CSS), HTML Template (vykreslení bloků až v době zavolání) a HTML Import (dovoluje vložit HTML, CSS a JS dokumenty do jiného dokumentu) [15].

Jak jsem již nastínil, Polymer, jako spousta jiných rámců, umožňuje definici vlastních HTML prvků, které lze libovolně modifikovat a propojovat pomocí *data-bindingu*. Vývojáři tak aplikují deklarativní způsob programování, jenž dále napomáhá k přehlednosti. Tento přístup taktéž umožňuje vytvářet prvky bez vizuálního významu, ty například slouží pro komunikaci s databází nebo práci s AJAXem.

U Polymeru se lze setkat s problémem podpory uvedených prvků Web Components u starších webových prohlížečů, kde například Firefox verze 45 nepodporuje, kromě HTML Templates, všechny tři výše uvedené komponenty. Chybějící podpora použitých komponent je řešena prostřednictvím knihovny Polyfill[8], která doplňuje kód nepodporovaný webovým prohlížečem o horší, ale v daném kontextu nejlepší možné, řešení k dosažení potřebných výsledků [32].

Aplikační rámec Polymer jsem zvolil pro jeho jednoduché, ale i přes to velmi efektivní, použití, s nímž mám už pozitivní předchozí zkušenost.

2.1.2 React

JavaScriptový aplikační rámec React byl založen v roce 2013 vývojáři sociální sítě Facebook a v dnešní době se těší vcelku velké oblibě, která ho staví mezi jeden z nejvíce populárních.

Jeho koncepce nabízí řešení některých typických problémů. Jedním z nich jsou těžkopádné změny v struktuře DOM, jež React řeší prostřednictvím paměti *Virtual DOM*, s její pomocí jsou vykreslovány pouze prvky, jichž se změna opravdu týká.

Dále lze mluvit o jednosměrném toku dat, který v kombinaci s knihovnou Redux vytváří MVC architekturu efektivně zabraňující smyčkám při propagaci změn [22].

2.1.3 AngularJS

Dalším aplikačním rámcem, který bezesporu patří mezi nejpopulárnější, je AngularJS založený v roce 2009 společností Google. Mezi jeho hlavní přednosti patří dvousměrný *data-binding*, jež řeší propojení stavů mezi modelem a pohledem v rámci MVC architektury, návrhový vzor zvaný *Dependency Injection* starající se o závislost mezi jednotlivými komponentami programu, dobrá testovatelnost, na niž je aplikační rámec zaměřen, a v neposlední řadě direktiva, která umožňuje rozšíření HTML o nové možnosti ať již formou nových elementů, nebo přidáváním atributů ke stávajícím elementům [19].

2.2 File API

File API je standardizovaný způsob jak přistupovat k lokálním souborům ve webovém prostředí. Zpřístupnění jednotlivých souborů je prostřednictvím formuláře, v němž jsou nabídnuty následující tři přístupy [28]:

[6] https://www.webcomponents.org/
[7] https://www.w3.org/
[8] https://polyfill.io

1. **File** – předmětem je jeden soubor poskytující informace pro čtení, což je hlavně jméno, velikost a odkaz na popisovač souboru.

2. **FileList** – předmětem je pole složené z jednotlivých souborů File, uplatní se při výběru více souborů formulářem s příznakem `multiple`.

3. **Blob** – prezentace souboru umožňující řezání po určitém počtu bajtů.

Vložený soubor není zpřístupněn prostřednictvím cesty v souborovém systému, ale přiřazeným popisovačem, jenž je platný po dobu platnosti relace, v níž byl soubor vytvořen. V případě zrušení anebo obnovení relace jsou odkazy na jednotlivé soubory zneplatněny. Standard je takto stanoven především z bezpečnostních důvodů, neboť by jinak každé aktivní webové aplikaci bylo umožněno přistupovat k libovolným souborům v systému. Webová aplikace tedy nemůže přistupovat k libovolným souborům, ale pouze k souborům vloženým uživatelem přes formulář. Tyto soubory jsou po zrušení nebo obnovení relace ztraceny.

Jakmile je získán odkaz na soubor, lze jej pomocí asynchronní akce nahrát do paměti RAM. Nabízí se čtyři možnosti čtení souboru, a to jako binární řetězec, textový řetězec, data URL anebo pole bufferu. Vzhledem k tomu, že operace načtení souboru do paměti není triviální, zvlášť pro objemné soubory, může trvat i v jednotkách desítek sekund. Na rychlosti je závislá i kapacita paměti RAM, kdy při přehlcení musí docházet k odkládání paměti na pevný disk, a tím se zpracování dále protahuje. File API nabízí sledování průběhu pomocí událostí, především zachycení postupného průběhu, ukončení anebo chyby zpracování [25].

2.3 Blob

Blob je zkratkou z anglického binary large object, což je datový typ blíže nespecifikovaných binárních dat. V oblasti jazyka JavaScript lze tento datový typ vytvořit a ukládat do něj potřebná data vcelku velkých objemů. Pro jeho ukládání slouží operační paměť RAM, na jejíž limity lze u většiny zařízení velmi brzo dosáhnout. Operační systém zařízení v této situaci z pravidla disponuje metodou odkládání operační paměti na disk.

Takto uložený blob v paměti RAM poskytuje rychlé provádění akce `slice`, která přečte zadaný úsek binárních dat a vrátí je jako nově alokovaný blob.

Dále je potřeba zmínit limit pro maximální velikost jednoho blobu, který z důvodu existence různých webových prohlížečů se značně liší. Postupným vývojem jsou limity navyšovány, v době psaní této diplomové práce jsou následující hodnoty u nejvíce populárních webových prohlížečů následující:

Webové prohlížeče postavené na jádru Chromium, tedy hlavně Chrome a Opera, odvíjí svůj limit z následujících podmínek:

- maximálně 2 GB pokud se jedná o 64 bitový systém a nejedná se o ChromeOS nebo Android
- maximálně jedna pětina celkové operační paměti
- maximálně polovina velikosti pevného disku u operačního systému ChromeOS
- maximálně dvacetina velikosti pevného disku u operačního sytému Android
- maximálně desetina velikosti pevného disku u ostatních systémů

Pro Mozilla Firefox verze 59 je stanoven limit 3 GB. V neposlední řadě je Safari, které disponuje limitem ve velikosti 1,8 GB. Tyto hodnoty nejsou nikde oficiálně uváděny a proto jejich stanovení proběhlo experimentálně.

2.4 FileSystem API

Jak již bylo řečeno v předchozí kapitole, webová aplikace nemůže přistupovat k souborům a manipulovat s nimi. Danou problematiku částečně řeší FileSystem API, který vývojáři nabízí v rámci aplikace vytvářet, číst, procházet a zapisovat soubory do bezpečně vymezené části souborového systému.

Pro využití tohoto API je zapotřebí zažádat o prostor voláním příslušné metody. Vyžádaný prostor se dělí na dva typy, a to dočasný a stálý. Dočasný prostor může být později smazán webovým prohlížečem bez vědomosti aplikace, například v případě nedostatku místa. Stálý prostor může být smazán pouze aplikací, která daný prostor vytvořila, a tedy je trvalý i po skončení a následném znovu obnovení relace. Pro stálý prostor je zapotřebí schválení v podobě dialogu, který musí uživatel potvrdit. Velikost prostoru v současné době není, až na samotnou kapacitu disku, nijak omezena [16].

FileSystem API přináší mnoho možností a využití. Bohužel v době psaní diplomové práce není příliš podporován. Podporu nabízí pouze webové prohlížeče s jádrem Chromium a vývojové skupiny ostatních webových prohlížečů zatím implementaci nepřislíbily. Podle statistik využití webových prohlížečů by k této technologii mělo mít přístup 69,16 %[9] uživatelů internetu.

2.5 Knihovna StreamSaver.js

Tato knihovna vznikla na základě potřeb ukládání objemných souborů, jejichž velikost přesahuje limity pro maximální velikost blobu anebo celkové velikosti paměti RAM. Knihovna asynchronně ukládá data přímo na paměť pevného disku za pomoci hlavních komponent, a to vcelku nového streams API, které umožňuje prostřednictvím JavaScriptu zpracovávat a vytvářet proudy dat přijímané prostřednictvím sítě, a Service Worker API, jež lze přirovnat k aplikačnímu proxy serveru využívaného pro akce vykonávané mezi webovou aplikací, prohlížečem a sítí.

Výše použité komponenty jsou využity k sofistikovanému ukládání souborů, které pracuje na základě vytvoření lokálního serveru prostřednictvím komponenty Service Worker. Mezi webovou aplikací a tímto serverem je vytvořen datový tok, do něhož webová aplikace postupně posílá data, jež potřebuje uložit. Server tato data přijímá a reprezentuje je jako soubor. Daný soubor začne webová aplikace stahovat prostřednictvím http žádosti. Jedná se o smyčku, v níž je díky komponentě Service Worker transformován datový tok na soubor. Takto lze zpracovat objemný datový tok postupným ukládáním na pevný disk zařízení.

V současné chvíli má v sobě knihovna StreamSaver zabudovaný v celku podstatný nedostatek v podobě časového limitu o velikosti 330 sekund[10], po kterém dojde k samovolnému restartování komponenty Service Worker. Přestože restart proběhne velmi rychle, dojde k přerušení stahovaného souboru a celý proces je tedy přerušen. Komunita ohledně této knihovny hovoří o ukončení v důsledku dojmu neaktivity z pohledu komponenty Service Worker. Na nedostatku komunita pracuje a lze očekávat vyřešení [8].

[9] https://caniuse.com//#search=FileSystem
[10] https://github.com/jimmywarting/StreamSaver.js/issues/39

Knihovna je podporována webovými prohlížeči založenými na jádru Chromium, kdy mezi nejznámější patří Google Chrome a Opera. Ostatní webové prohlížeče postrádají minimálně jednu ze dvou výše uvedených komponent [37]. Za zmínku však stojí Mozzila Firefox, jež postrádá komponentu streams API, která je již od verze 58 implementována ale implicitně odepřena. Lze ji aktivovat změnou příznaků `javascript.options.streams` a `dom.streams.enabled`.

2.6 Knihovna JsZip.js

Jedná se o knihovnu zpřístupňující akce pro vytváření, čtení a editaci komprimovaných souborů ve formátu ZIP prostřednictvím jazyka JavaScript. Takto jednoduše lze načíst soubory prostřednictvím File API nebo asynchronních dotazů a vytvořit z nich jeden komprimovaný soubor.

Lze vybrat jeden ze dvou datových typů, v němž bude soubor vygenerován, a to blob a arraybuffer. Z toho vyplývají již zmíněné limity z kapitoly 2.3. Výhodou je zajisté podpora této knihovny všemi známými webovými prohlížeči v jejich aktuální verzích.

2.7 Technologie serverové části

Většina webových aplikací disponuje serverovou části, jež se mimo jiné stará o správu, bezpečnost a integritu dat, výpočet některých operací a další úlohy. Klient zasílá data nebo požadavky na server, který provede zpracování a náležitou reakci. Logiku lze implementovat pomocí více programovacích jazyků, nejčastěji využívané bývají PHP, Python, Java, C# nebo i JavaScript.

S rychlým vývojem JavaScriptu se jeho použití čím dál více uplatňuje i v jiných oblastech než je *frontendu* webových aplikací, a proto současným trendem bývá použití JavaScriptu například právě v serverové části, kde je spuštěn pomocí Node.js[11]. JavaScript použitý na serveru umožňuje práci s vlákny v rámci *callbacků*, což napomáhá dobrému celkovému výkonu, manipulaci dat ve formátu JSON, která jsou nativně vestavěná, anebo použití jednoho jazyka v rámci celé aplikace.

Pro správu a operaci s daty jsou k dispozici databáze, podporující autorizaci a stanovení práv pro jednotlivé uživatele. Relativně novým přístupem v oblasti webového vývoje je použití samotné databáze jako celé serverové logiky. V takových případech, mimo standardní funkcionality databází, umožňují autentizaci a správu uživatelů. Pokud se vývojář rozhodne pro toto řešení, je zapotřebí klást velký důraz na integritní omezení pro jednotlivá data. Komunikace mezi aplikací a databází je zprostředkována pomocí REST API [21], případně prostřednictvím JavaScriptové knihovny.

Základním typem komunikace mezi *frontendem* a *backendem* je žádost HTTP, jejímž prostřednictvím je klientem na počátku stažena *frontendová* část aplikace. Další komunikace může nadále probíhat stejným způsobem. Tato komunikace je založena na žádosti klienta a odpovědi serveru a nikoliv naopak, což představuje hlavní problém protokolu, neboť server nemůže klientskou aplikaci kontaktovat v případě nějaké události. Řešením je pravidelné dotazování, takzvaný *polling*. V současné době se však uplatňuje naplno protokol WebSockets [11], který zajišťuje obousměrný komunikační kanál prostřednictvím protokolu TCP. *Backendová* a *fronendová* část je tak po celou dobo relace spojena kanálem, kde každá strana může bezproblémově zasílat data anebo indikovat odpojení, případně výpadek.

[11] https://nodejs.org/en/

2.7.1 SocketIo

Socket.IO[12] je knihovna umožňující komunikaci v reálném čase v rámci jednotlivých relací aplikace. Knihovna je rozdělena na dvě části, a to klientskou a serverovou část. Tyto dvě části mezi sebou ustanoví obousměrný kanál prostřednictvím WebSockets, díky kterému serverová část může koordinovat a synchronizovat všechny připojené klienty. Klientská část je implementována v jazyce javaScript a serverová část je dostupná v jazycích Java, C++, Swift nebo JavaScript, jenž je spouštěn již zmíněnou knihovnou Node.js [14]. V případě JavaScriptu jsou data obsloužena prostřednictvím *callbacků*, díky nimž jsou reakce rychlé a nevznikají žádné zbytečné prodlevy.

Knihovna se těší podpoře všemi webovými prohlížeči aktuální verze.

2.7.2 Databáze

Databáze je místo, kam se ukládají určitým způsobem organizovaná a strukturovaná data. Následný přístup, aktualizace a provádění operací nad těmito daty jsou rychlejší a jednodušší [35]. Nejpopulárnějšími databázemi jsou v současné době stále relační databáze využívající jazyk SQL, nicméně se čím dál více prosazují databáze podporující nerelační datový model, a to NoSQL databáze [2], jejich neznámějšími zástupci jsou MongoDB[13], CouchDB[14], Oracle NoSQL a další.

V případě relačního modelu jsou známy následující hlavní výhody. Data a datová integrita jsou snadno udržovatelná, nevznikají redundantní data a výpis lze lehce poupravit změnou SQL dotazu. Problém nastává při výskytu speciálních atributů, kdy může být zapotřebí pozměnit celé schéma databáze [27].

Typů NoSQL databází existuje více, například některé pracují na vztahu klíče a hodnoty, dále pak dokumentové, grafové a další. Díky těmto přístupům se nabízí techniky, jako například hashovací tabulky, které zajistí velkou rychlost zpracování. Uložené hodnoty mají např. podobu XML nebo JSON a jsou brány jako binární blíže nespecifikovaná data. Databáze se nesnaží uložená data chápat ale brát je jako celek, zpracování je provedeno až na aplikační vrstvě [27]. S tímto přístupem je připuštěna duplicita, nekonzistence a redundance dat. Dodržování modelu ACID (Atomicity, Consistency, Isolation, Durability) typického pro relační databáze je obtížné, a proto je dodržován model BASE (Basically Available, Soft State, Eventual consistency). Pro manipulaci s daty je stanoveno API.

2.7.3 MongoDB

Jedná se o multiplatformní dokumentovou NoSQL databázi vydanou v roce 2009, vyvíjenou v jazyce C++ [5]. MongoDB místo tradičních tabulek, charakteristických pro relační databáze, používá dokumenty ve formátu BSON, které představují obdobu široce známého formátu JSON. Záznamy jsou složeny z dvojice klíč-hodnota, kde hodnota může nabývat nejen základního typu, ale i pole hodnot či dokumentů, případně samotného dokumentu. Dokumenty lze takto do sebe vnořovat. MongoDB je pod licencí Open-source a zakládá si na vysokém výkonu, dostupnosti a automatické horizontální škálovatelnosti [12].

[12] https://socket.io/
[13] https://www.mongodb.com/
[14] http://couchdb.apache.org/

2.7.4 Parse Server

Návrh, implementace a zajištění bezpečnosti pro *backendové* části aplikací může být velmi náročné. Řešením mohou být dostupné platformy, jež nabízejí podporu při tvorbě těchto *backendů*.

Jedním, v současné době velmi populárním řešením, je platforma Firebase[15] vyvíjená a provozovaná společností Google. Ta disponuje mimo jiné NoSQL databází dostupnou přes REST API nebo klientskou knihovnu, autentizací přes účty Google, Facebook, Twitter a synchronizací dat mezi všemi připojenými klienty. Firebase nabízí opravdu velkou podporu při tvorbě a provozu aplikací, a proto se těší velké popularitě. Nicméně se jedná o placenou službu a není ji tedy možno provozovat na vlastních prostředcích, ale pouze pronajatých od společnosti Google.

Jednou z mnoha alternativ, které jsou dostupné pod licencí Open-source, je platforma Parse Server. Vychází z platformy Parse[16], jež byla do ledna roku 2017 [29] provozována a vyvíjena společností Facebook. Parse Server nelze svou funkcionalitou porovnávat s platformou Firebase, ale mezi open source o ní lze mluvit jako o jedné z nejlepších alternativ disponující silnou komunitou.

Parse Server využívá k ukládání dat MongoDB s funkcionalitou rozšířenou o refaktoring, indexaci, zálohu a obnovu. Dále je nabídnut propracovaný nástroj pro správu dat, ve kterém je umožněno lehce nastavit integritní omezení, správu dat anebo zasílání upozornění připojeným klientům. Pro aplikace založené na synchronizaci dat lze využít Live Quesries, jež zajistí upozornění aplikace na změnu sledovaných dat [31].

[15] https://firebase.google.com/
[16] http://parseplatform.org/

Kapitola 3

Peer-to-peer komunikace

Jedná se o decentralizovaný komunikační model, ve kterém má každá strana stejné možnosti a může zahájit komunikační relaci [6]. Protikladem je model klient-server, kde klient zahajuje komunikaci za účelem interakce se serverem. Server komunikaci nikdy nezahajuje, neboť neví, kde se klient nachází a nemůže s ním tedy navázat spojení.

Uzly v peer-to-peer sítích si jsou rovny, od čehož je odvozen i název. Tento model je využíván hlavně při přenosu dat mezi jednotlivými uzly, neboť oproti klient-server není zatížen třetí uzel, který by měl přenos zprostředkovat. Další výhodou je využití nejkratší a potenciálně nejvýhodnější cesty v síti, což má za následek její ušetření o zbytečnou zátěž a taktéž dosažení větší rychlosti přenosu.

Hlavní uplatnění si model peer-to-peer našel v torrentových sítích, kryptoměnách, IP telefonii, případně dalších službách založených na rychlé a objemné výměně dat. Některé aktivity peer-to-peer sítí jsou spojovány s nelegálními činnostmi, což může být důvodem pro jejich omezení v některých sítích z důvodu bezpečnostní politiky daného subjektu. Vzhledem k jejich obtížnému sledování a regulaci jsou mnohdy využívány k přenosu dat porušujících autorská práva, nebo jsou v rozporu se zákonem [26].

V současných sítích, které jsou stále převážně založeny na protokolu IPv4, nejsou peer-to-peer sítě zcela lehce implementovatelné vzhledem k nedostatku síťových adres v rámci protokolu IPv4. Zařízení jsou schována za překladem síťových adres a existuje tak více zařízení pod jednou síťovou adresou [30]. K řešení tohoto problému slouží signalizační servery, které vytvoří napříč všemi směrovači cestu v síti a ustanoví tak spojení k další komunikaci již bez pomoci jakékoliv třetí strany.

3.1 WebRTC

Tato kapitola se zabývá technologií WebRTC[1] (Web Real-Time Communication) vyvíjenou od roku 2013 [20] a která představuje klíčový prvek v mnou navržené webové aplikaci. Jedná se o Open-source projekt propagovaný hlavně společnostmi Google a Mozzila. Vývoj je veden W3C, což je mezinárodní konsorcium podílející se na vývoji webových standardů.

WebRTC poskytuje webovým a mobilním aplikacím možnost komunikace v reálném čase (RTC) bez zásuvných modulů prostřednictvím jednoduchým API [36]. Používá se především k hlasovým hovorům, videochatům a ke sdílení souborů. Knihovna je stále vyvíjena a probíhá adaptace některých webových prohlížečů, ale již ji lze z velké části bezproblémově

[1] https://webrtc.org/

použít. To mimo jiné potvrzuje i její rozšíření do mnoha známých webových aplikací, jako je například Hangouts, Facebook video chat a další [9].

3.1.1 Architektura

WebRTC je složeno z množiny protokolů, standardů a JavaScriptových API, jenž jako celek umožňují peer-to-peer propojení přenášející audio, video nebo datový přenos v rámci webového prohlížeče. Lze mluvit o třech hlavních API, která jsou implementována a to MediaStream, RTCPeerConnection a RTCDataChannel. MediaStream API umožňuje zachytávání a manipulaci s médii, jakou jsou zvuk a video ze vstupních zařízení mikrofonu a kamery. Výměna audio a video datových toků s podporou šifrování a správy šířky pásma je zahrnuto v rámci RTCPeerConnection API. Neméně významným je RTCDataChannel API implementující peer-to-peer komunikaci generických dat.

Z hlediska síťové architektury, kde mluvíme o peer-to-peer připojení mezi dvěma peery, lze stanovit dva druhy topologie, a to trojúhelník anebo lichoběžník. Topologie trojúhelník je získána v případě, že oba peery používají stejnou webovou aplikaci na stejném serveru, a topologie lichoběžník je získána, v případě propojení rozdílných aplikací na rozdílných serverech.

3.1.2 Signalizace

Tato kapitola je zaměřena na objasnění pojmu signalizace v rámci WebRTC. Hlavním úkolem signalizace je výměna metadat mezi peery pro zahájení a případnou koordinaci komunikace prostřednictvím stanoveného serveru. Obsahem vyměněných metadat bývají následující informace:

- Zprávy pro řízení relace sloužící k otevření a zavření spojení.
- Chybové zprávy.
- Metadata médií jako kodek a jeho nastavení, šířka pásma a typ média.
- Klíčová data pro vytvoření zabezpečeného připojení.
- Síťová data jako IP adresa a port.

Signalizace není součástí WebRTC vzhledem ke kompatibilitě s již existujícími systémy signalizace (XMPP, SIP), který vývojář může použít. Taktéž lze pro signalizaci použít vlastní implementaci například prostřednictvím WebSockets. Signalizační metody a protokoly taktéž nejsou součástí samotného standardu WebRTC a proto jsou popsány v protokolu JSEP (JavaScript Session Establishment Protocol) [18].

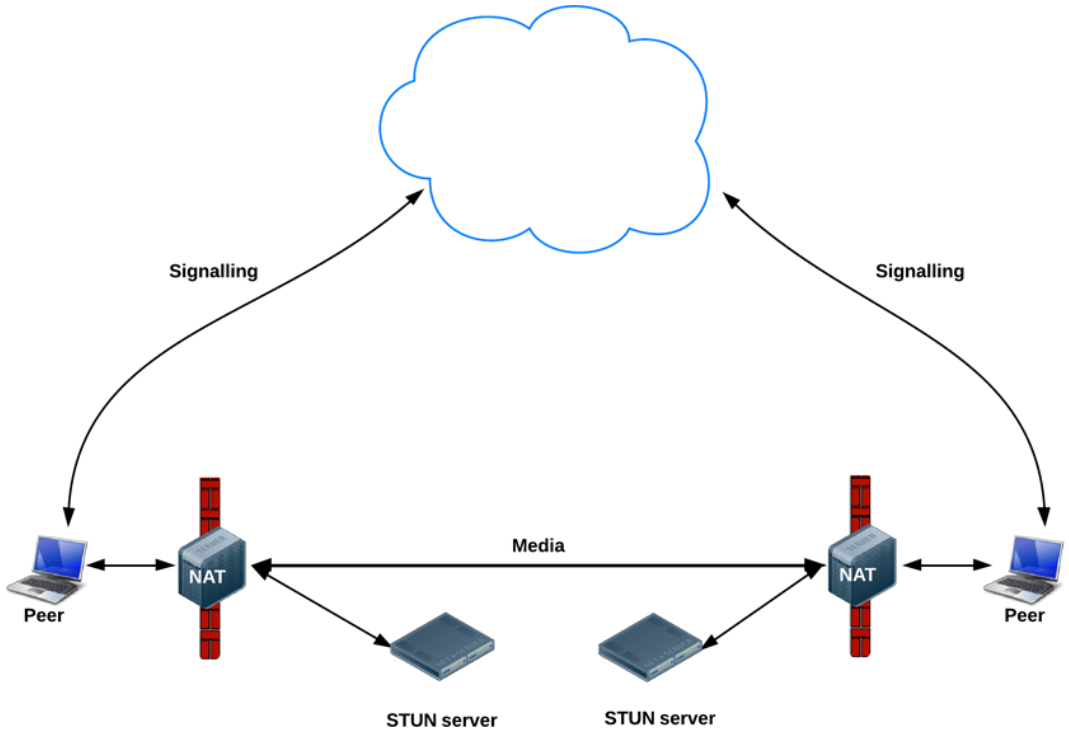
3.1.3 TURN a STUN servery

V reálné síti se příliš nesetkáváme se zařízeními, která mají unikátní adresu. Většina z nich se nachází za překladem síťových adres (NAT) a jejich operační systémy jsou obstarány antivirovými programy a firewallem blokujícím porty a protokoly.

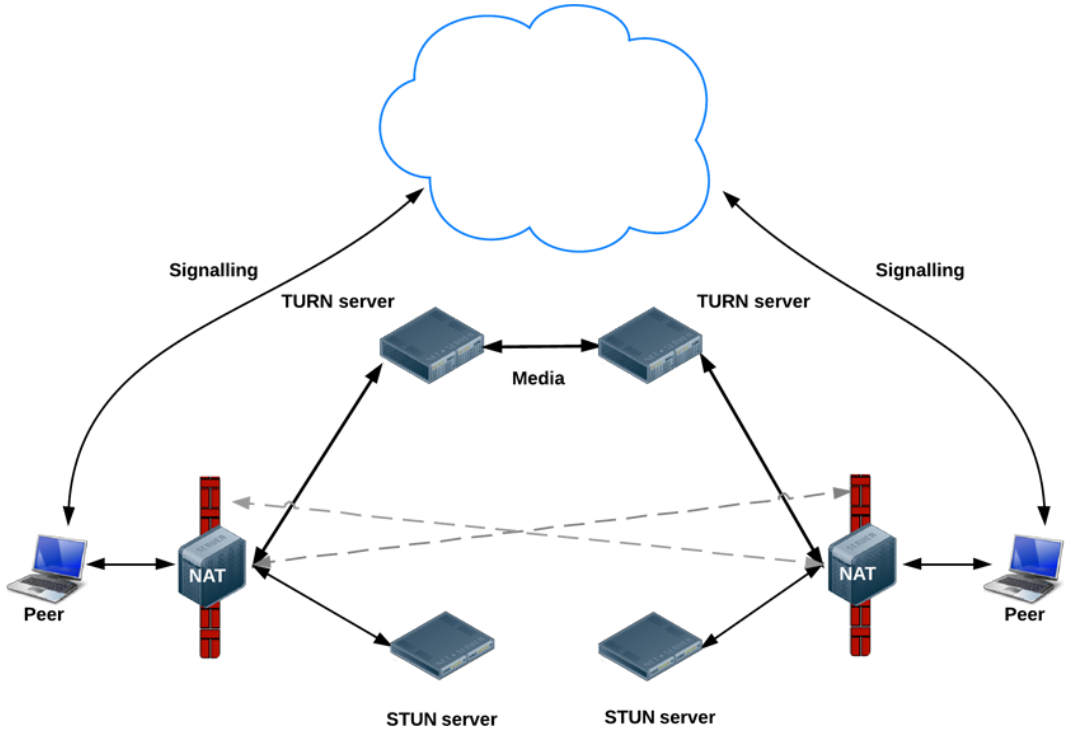
Aplikace používající technologii WebRTC využívají na základě těchto faktů aplikačního rámce ICE (Interactive Connectivity Establishment), který slouží k překonání složitosti sítí reálného světa. Aplikační rámec ICE se pokouší najít nejlepší cestu mezi dvěma spojujícími se peery a překonat průchod překladu síťových adres. K tomuto účelu slouží aplikačnímu

rámci ICE servery STUN (Session Traversal Utilities for NAT) a TURN (Traversal Using Relay NAT).

Hlavním úkolem STUN serveru je nalézt pro zadaný peer použitelný port v rámci unikátní IP, pod kterou peer vystupuje, k ustanovení veřejné přístupové cesty. Pokud je tento proces úspěšný u obou peerů, jsou následně přístupové cesty sděleny peerům navzájem, čímž obě strany znají přístup přes překlad adres, a tím může započnout komunikace. V případě neúspěchu v hledání přístupové cesty u serveru STUN se musí ustoupit od peer to peer komunikace. Přeposílání je následně zahájeno prostřednictvím TURN serveru [18].



Obrázek 3.1: Propojení dvou zařízení skrze překlad síťových adres, prostřednictvím serverům STUN. Zdroj [17].



Obrázek 3.2: Zprostředkování toku dat mezi dvěma zařízeními prostřednictvím serverů TURN. Zdroj [17].

3.1.4 Podpora

I přes spolupráci velkých webových prohlížečů jako jsou Google Chrome a Mozilla Firefox nelze mluvit o plné podpoře pro všechna zařízení uživatelů webu. Nicméně se ale jedná o vcelku malou skupinu, která se postupem času bude stávat menší. Dokonce v době psaní diplomové práce oznámila společnost Apple v blízké době vydání nové verze webového prohlížeče Safari s podporou WebRTC. Lze tedy mluvit o nedostupnosti služby pro uživatele Microsoft Explorer, který není nadále vyvíjen, a uživatele některých mobilních telefonů používajících výchozí webový prohlížeč s operačním systémem Android. Stále však mluvíme o podpoře přibližně 76 %[2] zařízení [10].

Lze předpokládat postupné zvyšování podílu zařízení, jež podporují WebRTC, vzhledem k plánované podpoře zbývajících mobilních zařízení a rozšíření nové verze webového prohlížeče Safari [13].

[2] https://caniuse.com/#search=WebRTC/

Kapitola 4

Existující řešení

Lze se setkat s již podobnými koncepty, které však disponují různými nedostatky, jejíchž vyřešení si nově navržená aplikace stanovuje za cíl. Tím by měl vzniknout nový, unikátní koncept, který předčí již existující řešení a nabídne tak uživatelům skvělý způsob sdílení dat ve skupině.

V následující kapitole budou existující řešení analyzována a rozebrána. Dojde tak na poukázání jejich nedostatků nebo naopak výhod, kterých by se nový koncept měl vyhnout nebo na druhou stranu inspirovat. Pro popis a porovnání jsem vybral pouze řešení, která pokládám za nejlepší, a tedy nejvíce konkurenční.

4.1 JustBeamIt

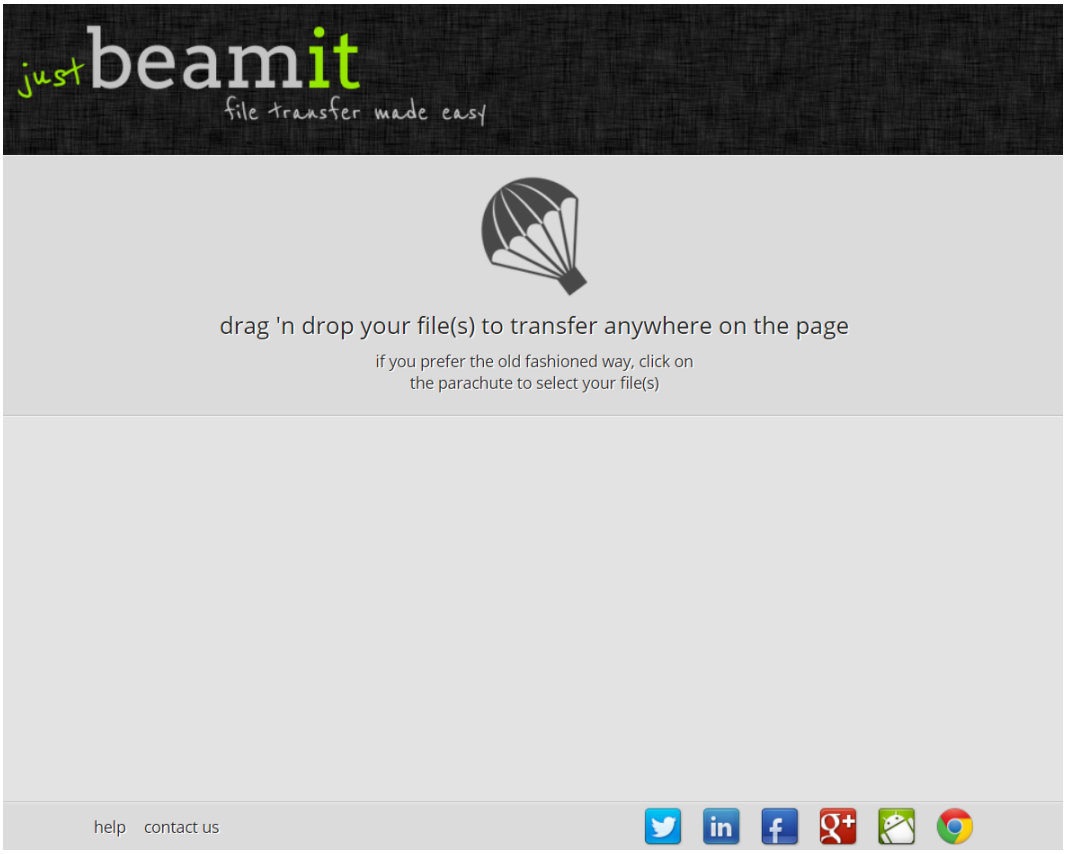
Jedna z nejjednodušších webových aplikací sdílejících základní principy je JustBeamIt[1]. Její hlavní charakteristikou je bezpochyby jednoduchost a intuitivnost. Prostor okna není přehlcen ovládacími prvky, a proto je zde místo pro popisný text, který je sladěn s vizuálním stylem a případnému uživateli objasní vše potřebné.

Jednoduché použití spočívá ve výběru souborů pomocí okna file manageru anebo akce drag and drop. Následně uživatel vyvolá akci vytvořením odkazu, který sdílí s příjemcem souboru. Odkaz je v podobě URL adresy a nebo QR kódu. Řešení je bezpochyby jednoduché a lehce použitelné, což se odráží v jeho popularitě na sociálních sítích.

Aplikace nedisponuje žádným limitem na přenášený soubor. Na druhou stranu disponuje limitem, který stanovuje přenos souboru pouze jednou přes vygenerovaný odkaz. Aby bylo plně aplikaci porozuměno, došlo k rozebrání komunikace, z které je patrné, že přenos není zprostředkován prostřednictvím WebRTC knihovny a nejde tedy ani o peer-to-peer přenos. Součástí aplikace je vlastní implementace serveru, který slouží k propojení odesílatele a příjemce, na základě které vznikají uvedené limity.

Oproti navrženému řešení zde postrádám sdílení ve skupině, zasílání celých složek, neomezené distribuce souborů a peer-to-peer přenos, který uživateli nabídne mnohem vyšší přenosové rychlosti, bez zbytečného zatížení síťových prvků [4].

[1] https://justbeamit.com/



Obrázek 4.1: Hlavní obrazovka aplikace JustBeamIt.

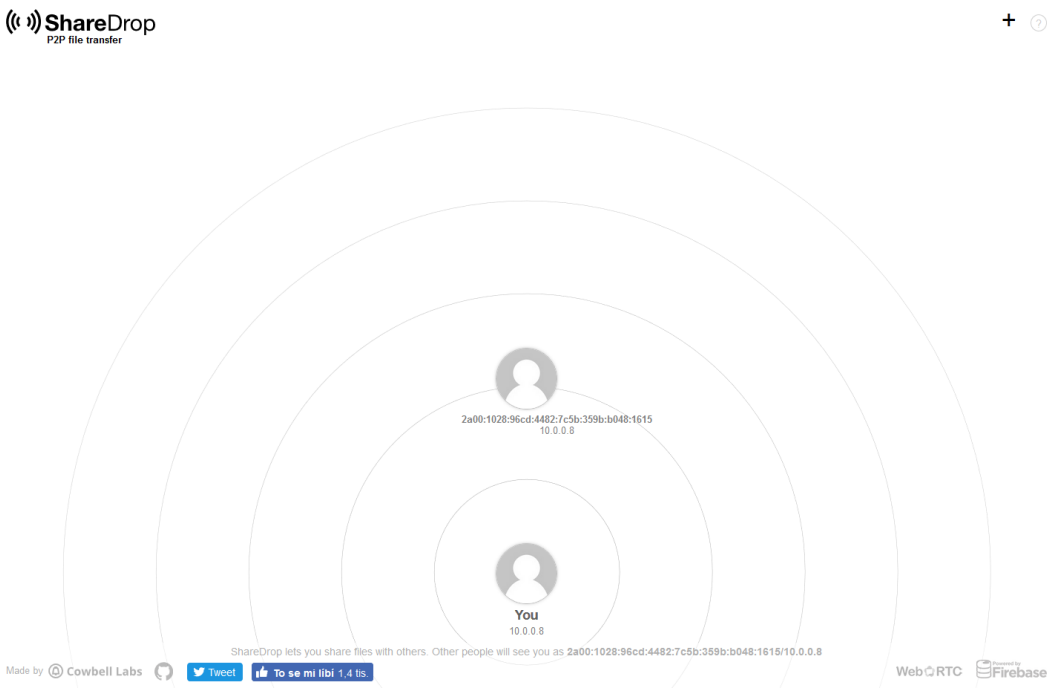
4.2 ShareDrop

Aplikace ShareDrop[2] je klonem služby AirDrop od společnosti Apple. Jak sami autoři přiznávají, je pouze přenesená do webového prostředí. Vyniká svým nadprůměrným vizuálním zpracováním, které lze z části připsat již zmíněné předloze od společnosti Apple. Uživatelé jsou znázorněni ikonkami v poli, kde je po kliknutí na uživatele zobrazen souborový manažer, v němž lze vybrat jeden soubor. Přijmutí souboru musí přijímající uživatel potvrdit.

Při prvním přístupu jsem věděl, co by aplikace měla přibližně dělat, ale i tak jsem měl problém v orientaci a použití aplikace. Odkaz pro sdílení místnosti není viditelný, nýbrž je schovaný pod ikonkou. Aplikace disponuje nápovědou, která uživatele v případě potřeby do jejího použití dostatečně zasvětí.

Za výhodu zde považuji skvělý vzhled a koncept místnosti, což lze uživatel považovat za atraktivní. Místnost je sdílena s libovolným množstvím účastníků, kteří si mezi sebou sdílí potřebný počet dokumentů. Nevýhodou této aplikace je posílání souborů po jednom a pouze jednomu účastníkovi. Dále je při zasílání potřeba postupná interakce obou stran podílejících se na přenosu. Účastníci si nemohou zasílat celé složky, stahovat více souborů naráz [34].

[2] https://www.sharedrop.io/



Obrázek 4.2: Hlavní obrazovka aplikace ShareDrop s dvěma aktivními uživateli.

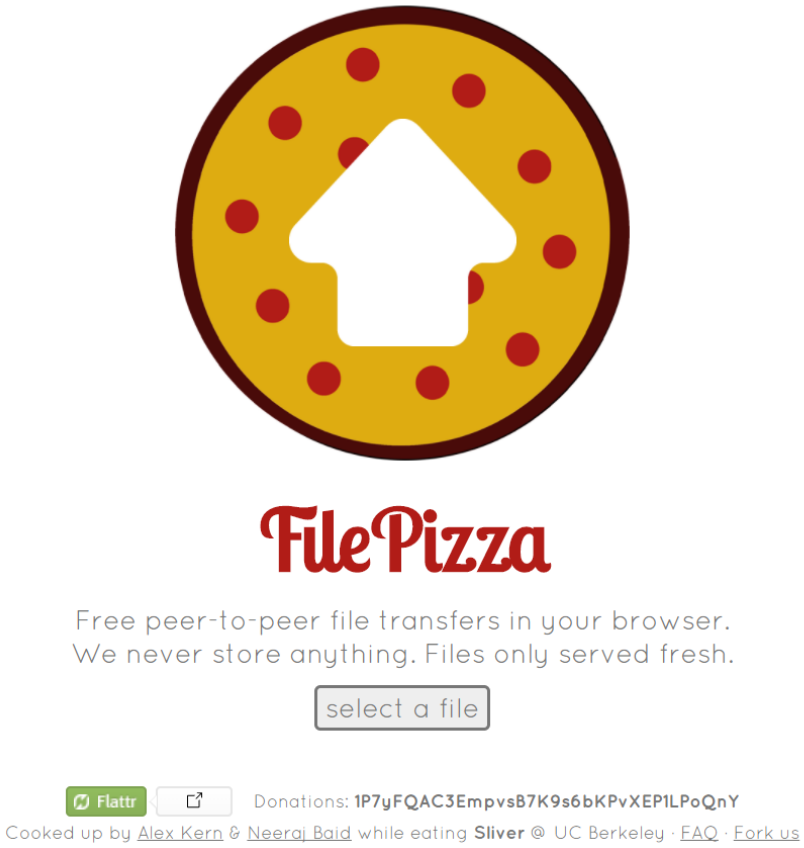
4.3 File Pizza

Jedním z nejatraktivnějších řešení shledávám aplikaci File Pizza[3], jejíž peer-to-peer přenos souborů je pojat zábavnou formou. Abstrakcí pro zasílaný soubor je pizza, která se distribuuje příjemci. Popisky, vzhled a nakonec i sdílená URL souboru jsou laděny s touto základní myšlenkou.

Použití je opět jednoduché. Uživatel vybírá soubor pomocí okna souborového manažera nebo akce drag and drop a následně sdílí vygenerovaný link. Vložený soubor musí aplikace zpracovat, přičemž předmětem operace je nejspíše načtení a rozložení souboru na části, jež se následně postupně zasílají. Toto zpracování trvá pro objemné soubory velmi dlouho a příliš objemné soubory nelze ani zpracovat, limit závisí na maximální velikosti blobu pro daný webový prohlížeč.

File pizza využívá knihovnu WebTorrent[4], díky které se na přenosu podílí všechna zařízení vlastnící soubor, což aplikaci dělá unikátní. Nevýhodu však představuje limit na velikost souboru. Uživatelé zde opět mohou postrádat sdílení více souborů, sdílení ve skupině a zasílání celých složek [24].

[3] https://file.pizza/
[4] https://webtorrent.io/



Obrázek 4.3: Hlavní obrazovka aplikace File Pizza.

4.4 Shrnutí

Analýzou existujících řešení byl zjištěn fakt, že neexistuje žádná služba, jež by navzájem spojovala skupinu uživatelů, kteří by si sdíleli své soubory. Této myšlence se částečně přiblížila služba ShareDrop, ale v rámci zobrazení nejsou uživatelé nijak odlišení. Není tedy jasné, kdo je kdo, a navíc při posílání souborů musí oba uživatelé přenos potvrdit.

Dalším nedostatkem většiny existujících řešení je limit na velikost přenášených souborů v prohlížečích mimo Google Chrome. Tyto limity mnohdy nejsou ani ošetřeny a stránka jednoduše zkolabuje. Webové řešení, které nedisponuje žádným limitem, je JustBeamIt. Důvodem absence limitu je způsob provedení přenosu, který není typu peer-to-peer, ale je založen na http žádosti na vzdálený server, což je velmi omezující vzhledem ke kapacitám serveru v podobě šířky sítě a výpočetního výkonu.

Poslední myšlenkou, kterou žádná služba nedisponuje, je zasílání celých složek. Mnohdy je zapotřebí zaslat celé adresáře skládající se z mnoha souborů, jejichž postupné zasílání je nemyslitelné a nereálné.

Kapitola 5

Návrh

Spread it out představuje moderní webovou aplikaci pro sdílení souborů ve skupině prostřednictvím peer-to-peer přenosu. Hlavním cílem je pro přenos souborů vytvořit takovou aplikaci, která není limitována použitou platformou a žádnými prerekvizitami. Jednotliví uživatelé tak lehce, efektivně a s minimálním úsilím sdílí soubory mezi sebou. Hlavní cíle, jež zároveň prezentují unikátnost navržené aplikace, spočívají v následujících bodech:

- **Peer-to-peer** – uvedená topologie přináší výhodu v podobě výměny dat pouze mezi zdrojovým a cílovým uzlem. Není tak zapotřebí vlastnit zdroje třetího uzlu k ukládání nebo přeposílání dat. Samotná rychlost je potom závislá na síti a prostředcích zdrojového a cílového zařízení.
- **Ve skupině** – hlavní výhodou oproti konkurenčním řešením je sdílení dat s více lidmi v podobě místnosti, v níž všichni uživatelé poskytnou svá data, která se volně rozšíří mezi ostatní uživatele, pokud o ně mají zájem. Z této myšlenky byl odvozen název aplikace, který je Spread it out.
- **Rychlost** – sdílení dat je zprostředkováno prostřednictvím místnosti, jejímž obsahem je odkaz na všechny soubory a účastníky pod jedním přístupovým URL. Data jsou v místnosti strukturována a je evidován jejich stav. Uživatel tak může rychle a efektivně přistupovat k mnoha souborům pod jedním URL. Samotný přenos se skládá z uploadu na straně odesílatele a downloadu na straně příjemce, který probíhají v rámci přenosu paralelně.
- **Bezpečnost** – oproti většině nástrojů pro sdílení dat na internetu zde není využívána třetí strana, které je svěřena důvěra. Samotný přenos je šifrován, čímž je eliminováno odposlouchávání dat. Místnosti lze zabezpečit heslem pro eliminaci nežádoucích hostů.

5.1 Místnosti a sdílení

Základem aplikace jsou místnosti, které představují oddělený prostor, v němž vybraní uživatelé sdílí data mezi sebou navzájem. Identifikace místnosti je dána jménem, která je pro danou místnost unikátní. K místnosti lze přistoupit z úvodního okna, prostřednictvím formuláře a nebo odkazem URL, který slouží jako prostředek pro přizvání dalších účastníků. Název vepsaný do formuláře aplikace zpracuje a uživatel je připuštěn do již existující místnosti nebo je na základě názvu místnost vytvořena. Není tak potřeba se starat o to, jestli místnost pod daným názvem již existuje. K přístupu a vytvoření stačí jeden formulář.

K sdílení místností lze přistupovat dvěma různými přístupy. První přístup předpokládá existenci místností nesoucí názvy různých oblíbených termínů, jako jsou názvy velkých aplikací, komunit, organizací a podobně. Tyto místnosti nejspíše neposkytnou příliš velkou autonomii, naopak lze předpokládat již spoustu připojených účastníků, kteří k danému tématu budou sdílet data. Druhým přístupem je opravdu autonomní místnost jen pro určitou komunitu, která mezi sebou chce sdílet data bez rizika náhodného hosta. V této situaci se zakládající uživatel může pokusit být kreativní a vytvořit místnost s unikátním názvem, avšak je zde nadále riziko případného hosta. Proto aplikace nabídne vytvořit místnost zajištěnou heslem, jehož hodnotu bude zapotřebí znát k úspěšnému vstupu.

5.2 Zachování relace

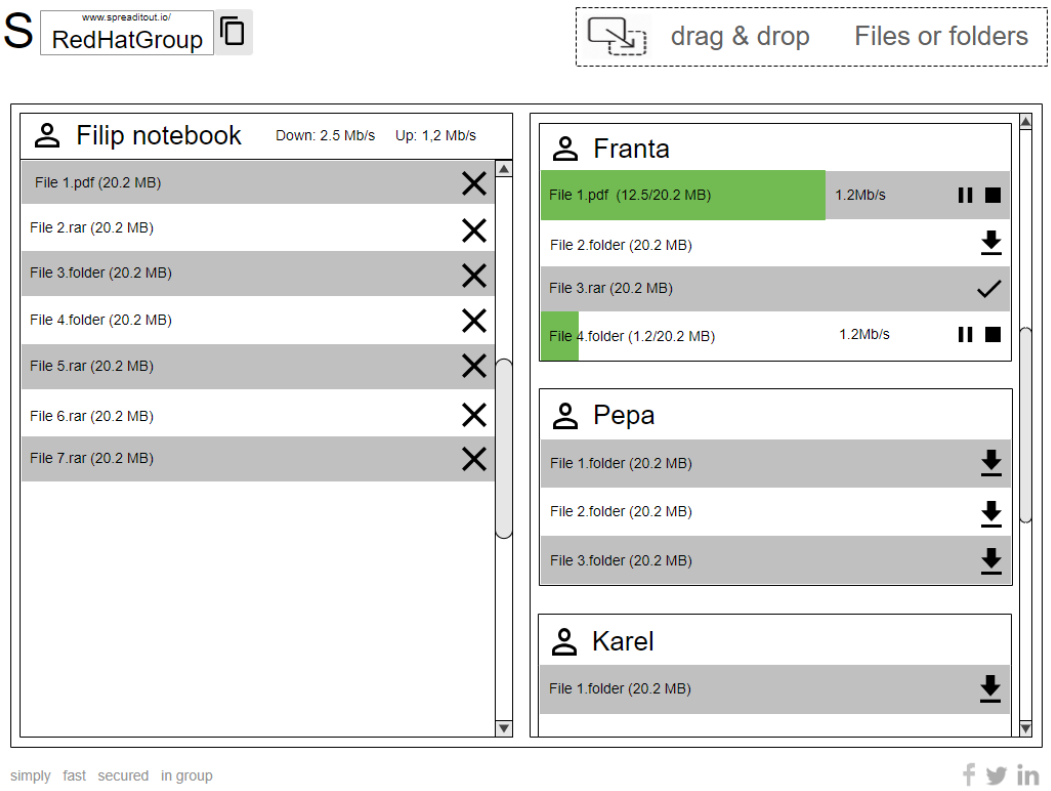
Uživatel do místnosti vkládá soubory, jež jsou zpřístupněny všem ostatním účastníkům a bylo by ideální je zachovat i pří vypnutí a následném zapnutí aplikace, což na základě omezení, která jsem rozebíral v kapitole 2.2, není možné. Podporu lze nabídnout v podobě dialogu upozorňujícího na zavírající se relaci a ztrátu stavu. Toto chování představuje i obranu proti náhodnému zavření aplikace.

Dalším konfliktem, který je potřeba ošetřit, je otevření jedné místnosti ve více oknech, čímž může docházet k rozptýlení řízení. Tím by uživatel ztratil povědomí o tom, do jakého okna nahrál jaké soubory. Z tohoto důvodu aplikace nedovolí otevření jedné místnosti v rámci více oken a přístup k již otevřené místnosti. Znovu otevření místnosti bude stornováno a uživatel o této akci je obeznámen prostřednictvím dialogu.

5.3 Zobrazení dat

Místnosti mohou nabývat velkých rozměrů jak po stránce počtu účastníků, tak i celkového množství zpřístupněných souborů. Proto je zapotřebí potencionálně velké množství dat vhodně prezentovat.

Samotná prezentace dat je zaměřená na znázornění vztahu jednotlivých účastníků a jejich souborů. Každý aktivní účastník je reprezentován vyhrazeným prostorem, jenž bude obsahovat název uživatele a výčet všech jeho souborů. U každého souboru se uvádí název, velikost a stav, tedy jestli je již stažen, nestažen nebo právě stahován. Dále je opatřen nabídkou akcí pro změnu jeho stavu, případně koordinaci stahování. Stahovaný soubor je obohacen o rychlost stahování a jeho průběh. Speciální prostor je vyhrazen i pro koordinaci dat v rámci otevřené relace, v níž se uvádí jméno uživatele, které lze libovolně měnit, rychlost celkového stahování, odesílání dat a v neposlední řadě zde uživatel najde výčet všech svých souborů s možností jejich odebrání. Při větším počtu uživatelů a jejich souborů nelze vše zobrazovat v jednom okně, proto je zobrazena vždy část uživatelů a část souborů, zbytek je dostupný prostřednictvím rolování. Tento koncept je znázorněn na obrázku 5.1.



Obrázek 5.1: Zobrazený návrh stránky prezentující místnost, jejímž obsahem jsou uživatelé se svými soubory. V levé části lze nalézt relaci místního uživatele, která disponuje jménem uživatele, sledováním rychlosti přenosu a výčtem vložených souborů. V pravé části je znázorněn výčet uživatelů s jejich soubory. Zobrazena je jen část uživatelů, ke zbytku lze přistoupit prostřednictvím rolování. Soubory, jež uživatelé vlastní, disponují stavem a operacemi, které lze nad nimi provádět. V hlavičce stránky jsou znázorněny operace pro sdílení místnosti a vložení souborů či složek.

Kapitola 6

Implementace

Tato kapitola je věnována implementaci již navržené aplikace z předchozí kapitoly. V první části implementace je hovořeno o grafickém zpracování a chování uživatelského rozhraní. Dále je rozebrána nejen problematika zpracování souborů a složek na straně odesílatele, ale také zpracování datového toku na straně příjemce. Poté následuje detailní popis serverové části, jež se stará o synchronizaci připojených relací, ukládání dat a v neposlední řadě bezpečnost celé aplikace. Závěr je věnován dalšímu rozvoji celé aplikace.

6.1 Uživatelské rozhraní

Úvodem této kapitoly, která se bude zabývat uživatelským rozhraním, bych rád zmínil základní myšlenku celého uživatelského rozhraní implementované aplikace. Mnoho existujících aplikací je vzhledem, ovládáním a chováním implementováno podle předloh v podobě vizuálního jazyka materiálního designu, což je systém jednotného stylu, chování a interakcí splňujících stanovenou množinu principů. Použití těchto předloh je z mého pohledu bezesporu dobře funkční a vcelku méně náročné oproti navrhování vlastního konceptu. Cílem mnou vytvořeného grafického návrhu bylo vytvořit něco originálně hravého, něco, co vyvolá v návštěvníkovi emoce, jež by mohly zvýšit šanci si aplikaci lépe zapamatovat a přemýšlet o ní. Tím jsem však nechtěl uživatele připravit o intuitivnost a srozumitelnost, které jsou prioritami pro vytvořenou aplikaci a zároveň základní důležitou podmínkou pro úspěšné rozšíření mezi širokou veřejností.

6.1.1 Správa klientských závislostí

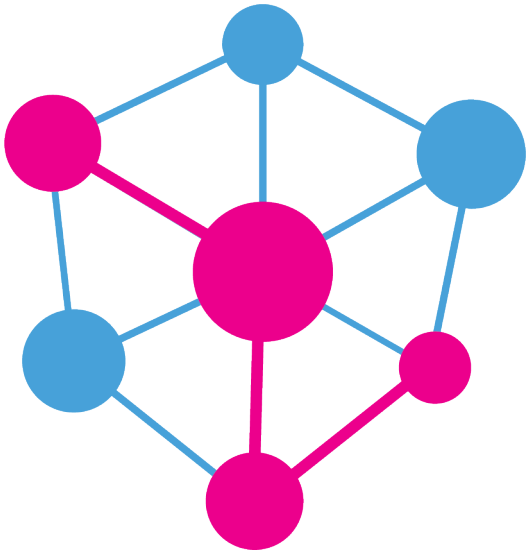
Na začátek je zapotřebí zmínit nástroje pro správu knihoven třetích stran, které byly v rámci implementace použity. Pro knihovny podporující vzhled, chování a architekturu uživatelského rozhraní byl použit správce balíčku Bower, jehož konfigurační soubor `bower.json` obsahuje všechny nainstalované balíčky. V rámci tohoto nástroje byly spravovány, mimo v dalších kapitolách uvedených, knihovna Polymer a její závislosti, JavaScriptová knihovna jQuery pro manipulaci s prvky DOM nebo knihovna Redux.

Knihovny pro síťovou komunikaci SocketIo, Parse, Simple-peer a jejich závislosti byly spravovány prostřednictvím správce balíčku npm, v němž byly dostupnější.

6.1.2 Grafický vizuální styl

Grafický vizuální styl je první věc, s níž se uživatel při přístupu na webovou stránku seznámí a jež je základem pro utvoření prvního názoru. S tímto dojmem jde ruku v ruce i míra důvěry, kterou bude uživatel k dané službě pociťovat. Z tohoto důvodu je nezbytné věnovat této části značnou pozornost. Pro běžného uživatele je tento podnět velmi důležitý, neboť jiným záležitostem, jako jsou technologie, chování a složitost provedení, nemusí až tolik rozumět.

V rámci aplikace jsem se pokusil důkladně promyslet vizuální styl a snažil jsem se ho opřít o technologii, která je v aplikaci použita. Prvním odkazem jsou použité barvy a samotné logo. Nejčastěji jsou použity dvě základní barvy, a to odstín modré s hodnotou rgb 75, 161, 217 a odstín růžové s hodnotou rgb 236, 0, 140. Tyto dvě barvy odkazují na používané barvy pro symboly nahrávání a stahování, kdy prvky pro stahování jsou modré a prvky pro nahrávání červené. Přímá kombinace modré a červené mi připadala až moc konvenční, a proto jsou zvoleny tyto dvě barevné alternativy. Dále je zde logo viditelné na obrázku 6.1, které je krom již zmíněných barev složeno z navzájem propojených koulí, přičemž každá koule reprezentuje rovnocennou entitu, což odkazuje na princip peer-to-peer propojení. Složením barev a koulí reprezentujících logo odkazuje na koncept rovnocenných entit, v němž se některé nacházejí ve stavu odesílání a jiné ve stavu přijímání dat.



Obrázek 6.1: Logo reprezentující aplikaci, jež vyjadřuje vzájemné propojení uzlů a jejich rovnocennost.

Dalším odkazem na princip peer-to-peer je samotné velmi rozsáhlé pozadí, které se prolíná celou aplikací. Takto rozsáhlé pozadí bylo použito z důvodu maximálního využití celého prostoru okna poskytnutého webovým prohlížečem, a tedy obrazovkou zařízení. Pozadí disponuje opět mnoha navzájem propojenými koulemi, jež se v prostoru různě prolínají a prezentují tak rozmanitost propojení jednotlivých entit v prostředí velmi rozsáhlých sítí dnešního světa. Na úvodní obrazovce se lze setkat pouze s modrými koulemi (lze vidět na obrázku 6.2), které jsou více neutrální a méně rušivé. Avšak v rámci vytvořené místnosti je pozadí rozděleno na část s modře a na část s růžově zbarvenými koulemi, což lze vidět na obrázku 6.5. Modré se nacházejí v části, v níž se vyskytují data a soubory místního uživatele. Barva tak odkazuje na informace, které má uživatel u sebe. Zatímco v části růžové jsou

data ostatních uživatelů, a tedy data, jež jsou od něj odloučena a zároveň jsou to entity, jimž bude svá data nahrávat.

Právě uvedený barevný a tvarový koncept je prolnut celým uživatelským rozhraním. Snažil jsem se využít pouze tyto dvě barvy v kombinaci s černou barvou pro některé texty a stupni šedi pro neaktivní či nedostupné prvky.

Aplikace disponuje značným množstvím tlačítek pro ovládání a koordinaci, jež bylo možno koncipovat jako textová anebo ikonová. Výhodou textových tlačítek je pro většinu uživatelů zajisté jejich jasnější pochopení na první pohled, jejich nevýhodou je ale zbytečný prostor, který zabírají, zvlášť v případě mobilních zařízení, u nichž se hodí každý pixel. Vzhledem k charakteru aplikace, jejíž místnosti mohou narůstat do velmi vysokého počtu uživatelů a souborů, jsem se většinou rozhodl pro použití ikonových tlačítek. Ikonová tlačítka mohou být často nesrozumitelná, což je jejich nevýhoda, a proto je potřeba implementovat kompenzaci, čímž se zabývám v následující kapitole.

V aplikaci se lze setkat s velkým výskytem vertikálního rolování, které jsem se snažil sladit s celkovým vizuálním stylem, a to hlavně s rolovacími posuvníky. Vzhledem k tomu, že některé webové prohlížeče nepodporují úpravu těchto posuvníků, byla použita knihovna perfect-scrollbar[1], jež zachovává funkcionalitu rolování na základě kaskádových stylů, ale vykresluje vlastní posuvník, čímž bylo docíleno sjednocení v rámci všech zařízení bez ohledu na použitý webový prohlížeč.

6.1.3 Navigace a upozornění

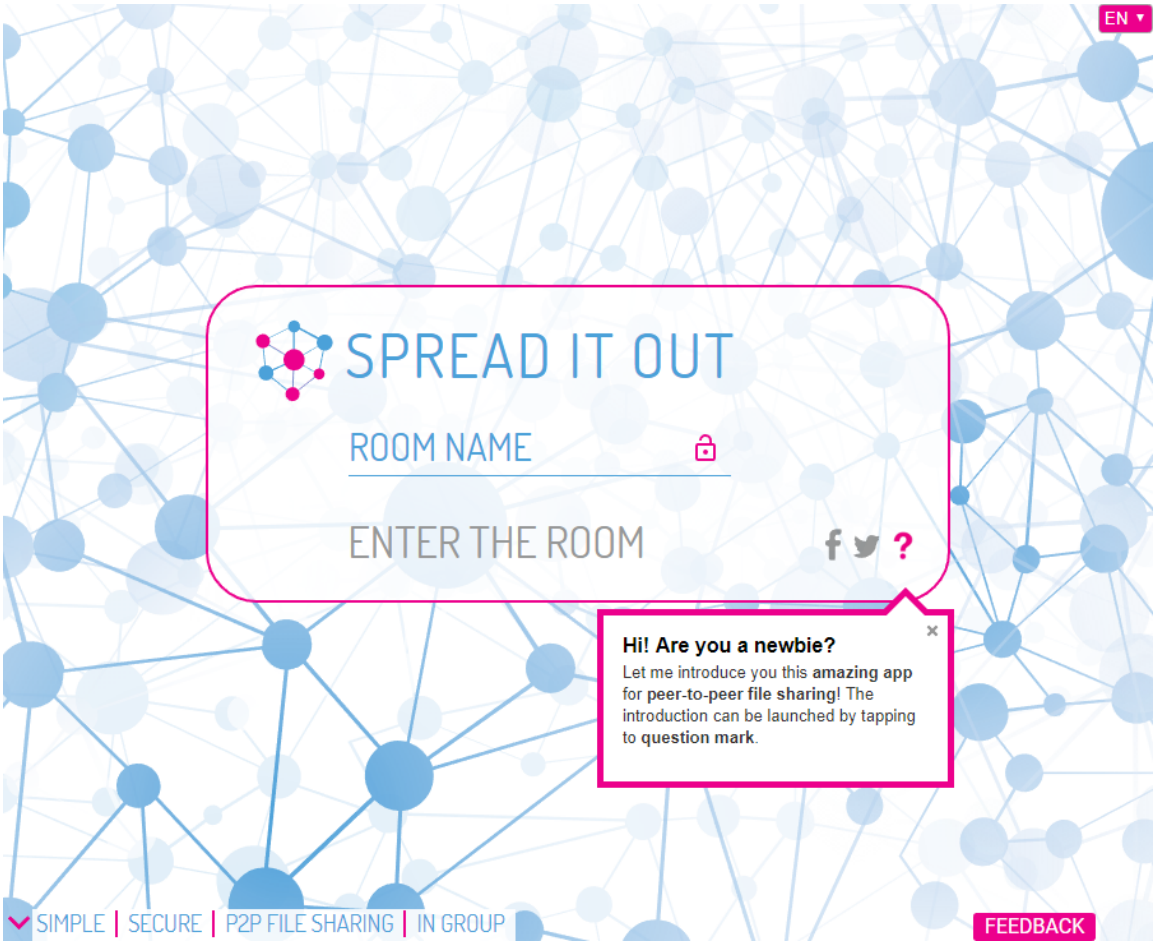
Vzhledem ke spoustě akcí a podmínek, kterými aplikace disponuje, je nezbytné vést jakýsi dialog s uživatelem, informovat jej o všech akcích, jež jsou provedeny, a případně upozornit nebo navést na zdárné provedení akce. Z tohoto důvodu aplikace disponuje čtyřmi základními prvky pro komunikaci s uživatelem.

Prvním a základním prvkem jsou vlastní nápovědní dialogy, jimiž disponuje každé ikonové tlačítko, jehož význam nemusí být uživateli vždy úplně zřejmý, a proto je při najetí na toto tlačítko uživatel seznámen s jeho významem.

Druhým, v celku populárním, prvkem jsou tak zvané oznámení *Toast* nebo *Snackbar*, což jsou vyskakující pruhy zpravidla v dolní části obrazovky. Tato oznámení jsou zpravidla využita pro upozornění o dokončení, případně nezdaru zadaných akcí a nebo obecné změně stavu aplikace.

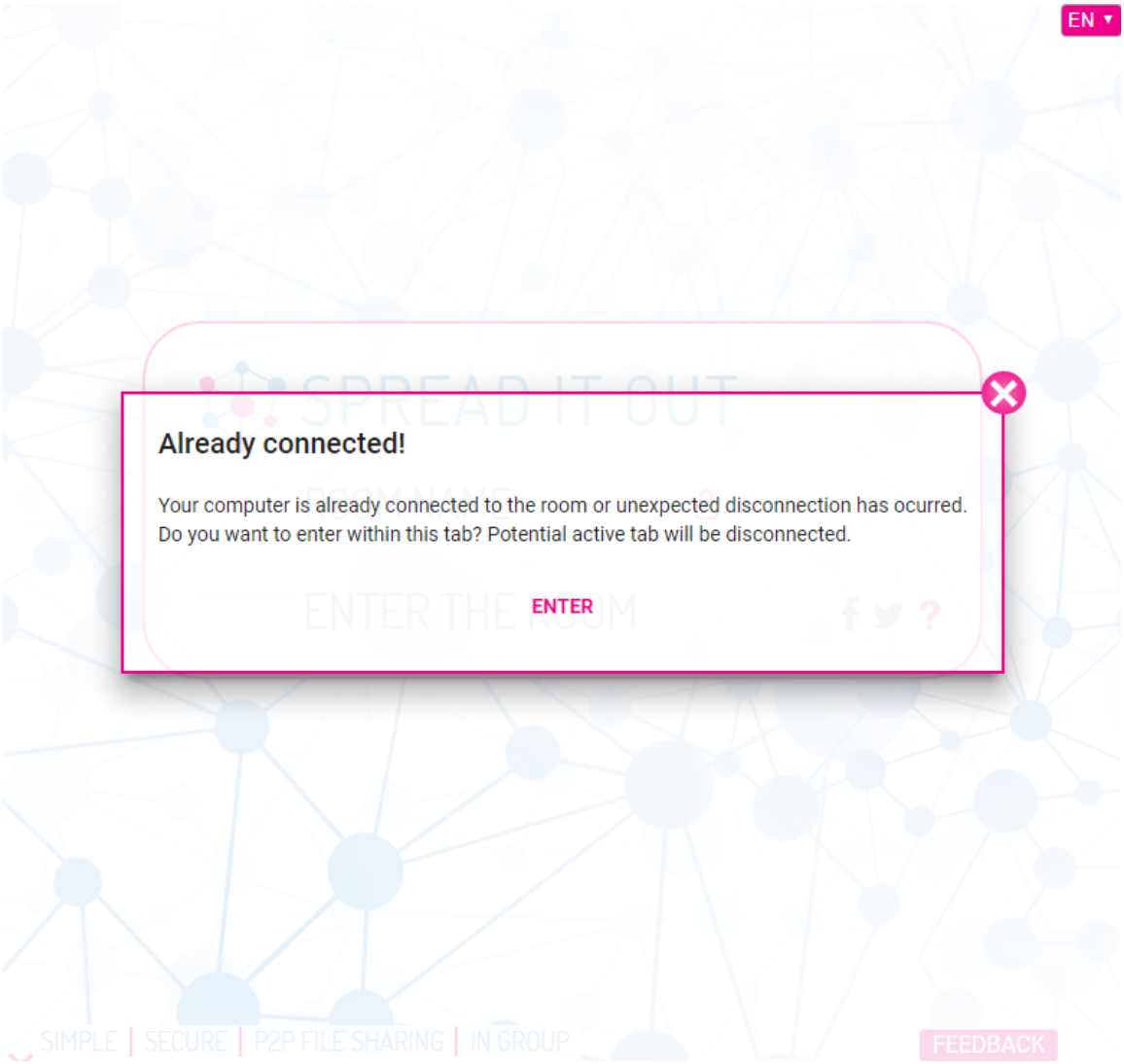
Dalším prvkem jsou vyskakující dialogy u určených prvků aplikace. Tyto dialogy jsou větších rozměrů a díky tomu jsou vhodné pro rozsáhlejší texty, v nichž je zapotřebí uživateli něco lépe přiblížit, případně popsat. Tyto dialogy byly použity i v podobě postupně navazující prohlídky, při které postupně vyskakují u zvolených elementů, tím pak přibližují jejich význam a funkcionalitu. Jejich použití lze vidět na obrázku 6.2. K této funkcionalitě byla použita JavaScriptovou knihovnu Hopscotch[2] vyvinutou sociální sítí LinkedIn[3].

[1] https://utatti.github.io/perfect-scrollbar/
[2] http://linkedin.github.io/hopscotch/
[3] https://www.linkedin.com/



Obrázek 6.2: Úvodní obrazovka aplikace, s doprovodným dialogem upozorňující nového návštěvníka na funkci průvodce aplikací.

Poslední prvek představují dialogová okna, která vyžadují od uživatele maximální prioritu, a proto znemožňují další práci s aplikací, dokud není dialog vyřešen. Tyto dialogy jsou většinou využity pro provedení zásadních akcí, jako je například odejití z místnosti anebo při konfliktu s již aktivním oknem ve stejné místnosti. Použití tohoto prvku lze vidět na obrázku 6.3.



Obrázek 6.3: Dialog upozorňující uživatele na vstup do místnosti, která je již pravděpodobně v rámci webového prohlížeče již jednou otevřena.

6.1.4 Úvodní stránka

První přístup uživatele směřuje na úvodní stranu webové aplikace a prvotním úkolem je neznalému návštěvníkovi přiblížit princip celého konceptu a seznámit ho s argumenty, proč by pro něj aplikace mohla být atraktivní. Z tohoto důvodu byla implementována decentní úvodní animaci k seznámení uživatele s hlavními myšlenkami aplikace. Animace probíhá místo ovládacích prvků, jež jsou po jejím provedení zobrazeny. Dalším prvkem podpory pro snadnější zorientování uživatele je vyskakovací dialog vybízející ke spuštění prohlídky v rámci úvodní stránky, jak lze vidět na obrázku 6.2.

Prohlídka v podobě postupně na sebe navazujících vyskakujících dialogů ještě blíže seznámí uživatele s celým konceptem a ovládacími prvky. Třetím stupněm seznámení může uživatel projít v podobě patičky, která obsahuje základní hesla, a postupným proklikáváním se hesly jsou zobrazovány obrazovky s rozsáhlými dialogy detailně popisujícími myšlenku

celého konceptu. Takto je myšlenka aplikace postupně servírována uživateli od obecnější specifikace ke konkrétnější.

Vzhledem k tomu, že samotná animace a vyskakující dialogy mohou být s opakovanou návštěvou služby nežádoucí a otravné, aplikace ukládá počet návštěv do perzistentní lokální paměti, tak zvané *cookies*. Při více jak třech návštěvách nejsou animace ani dialog dále zobrazovány.

Hlavním obsahem úvodní stránky je vstupní pole pro zadání jména místnosti, bez jehož vyplnění nelze vstoupit, a tím tedy tlačítko pro vstup není aktivní. Jakmile je zadáno jméno, lze taktéž pomocí ikony zámku rozbalit další vstupní pole pro vložení hesla v rámci vytvářené místnosti. Potom následuje samotný vstup do dané místnosti.

Z pohledu uživatele není schválně rozlišeno, jestli uživatel do místnosti vstupuje anebo ji vytváří. Uživatel zadá jméno a do místnosti vstoupí. Aplikace řeší situaci následovně. Pokud místnost doposud neexistuje, vytvoří ji a následně uživatel vstoupí, jinak uživatel pouze vstoupí do již existující místnosti. Konflikt nastává v případě zadávání hesla. Pokud by bylo možné přidat heslo k libovolné místnosti, mohou si je uživatelé navzájem heslovat a docházelo by tak ke konfliktům. Z tohoto důvodu lze zvolit heslo jen v případě, že místnost doposud neexistuje, a to je důvodem, proč rozbalení vstupního pole doprovází animace načítání. V průběhu animace je proveden dotaz na server a stav ověřen. Pokud místnost neexistuje, lze stanovit heslo místnosti.

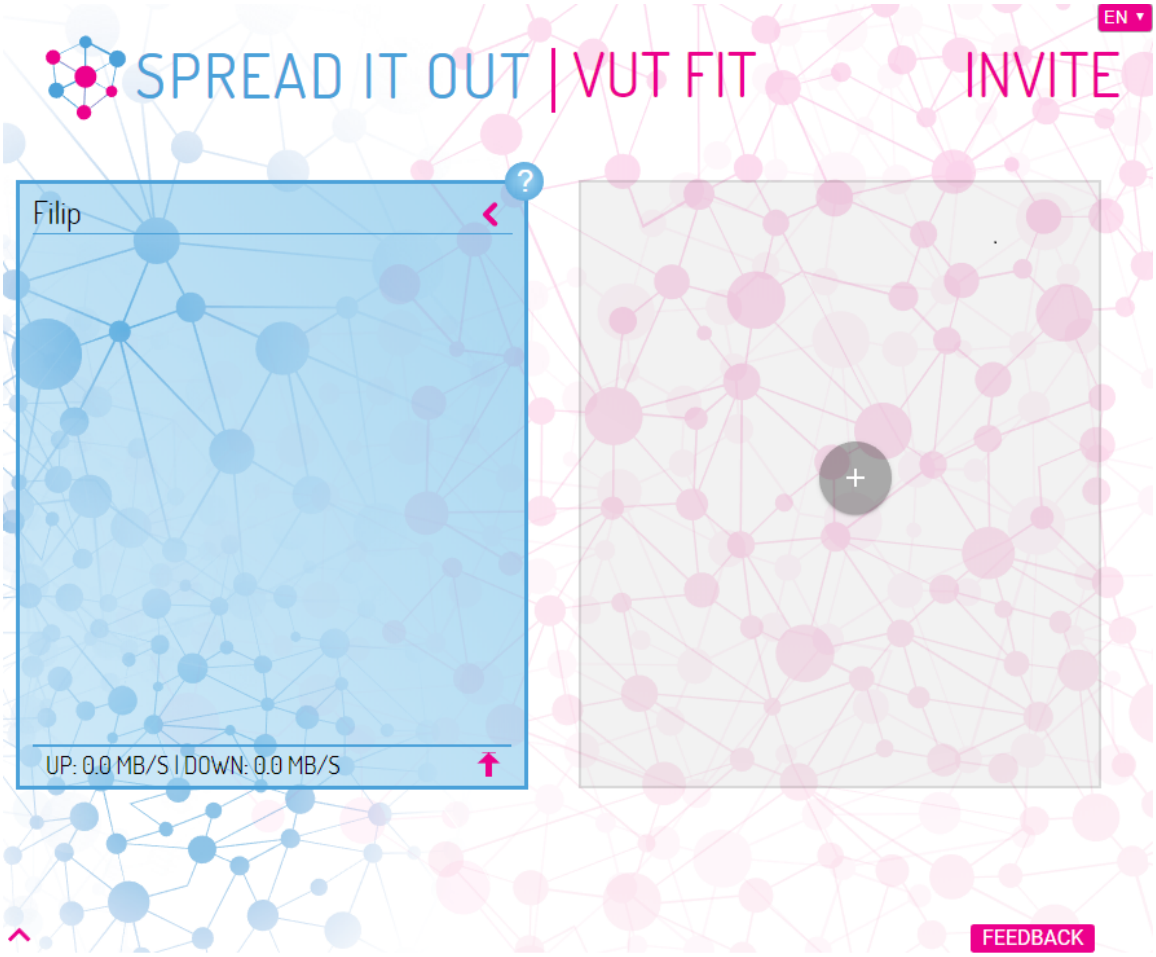
Následuje vstup do místnosti, v jehož průběhu se mohou vyskytnout dvě situace. První situace je vyzvání k zadání hesla, pokud je místnost pod heslem, což je implementováno postupnou interakcí uživatelské aplikace a serveru. Uživatel zadá jméno místnosti, na nějž server odpoví žádostí o heslo. Uživatel provede stejný dotaz, ale obohacen zadaným heslem.

Druhé situace v rámci vstupu do místnosti lze docílit v případě vstupu druhé záložky do stejné místnosti. Aplikace je záměrně implementována pouze pro jednu aktivní záložku v rámci jedné místnosti. Cílem je udržet veškerá data a kontrolu v jednom okně, neboť pro samotného uživatele musí být matoucí, pokud jsou data a ovládání roztříštěna napříč více okny, která jsou právě aktivní. Takto samotná aplikace navádí uživatele ke správnému a smysluplnějšímu chování. Pokud je tedy detekována kolize, uživatel obdrží dialog informující o dané situaci a nabídne zrušení vstupu nebo pokračování pod podmínkou, že původní aktivní okno bude odpojeno z místnosti. Pokud se tak stane, vyhozené okno z místnosti je přepnuto na úvodní obrazovku s dialogem oznamujícím odpojení z místnosti z důvodu vyhození jiným oknem.

6.1.5 Stránka místnosti

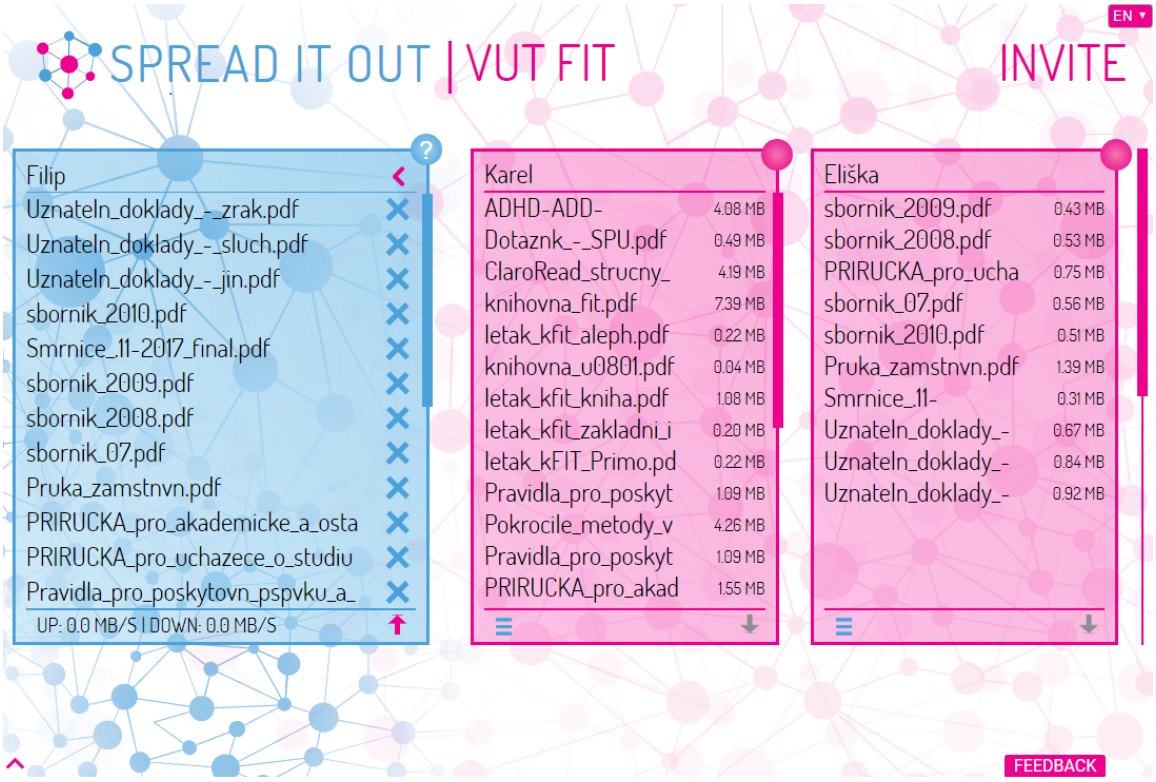
Vstupem do místnosti je provedena změna obrazovky, jejíž vzhled lze vidět na obrázku 6.4. Pokud se uživatel vyskytl v místnosti poprvé, je vyzván k zadání jména, kterým se bude prezentovat ostatním členům místnosti. Pro druhý a další vstup je jméno již zapamatováno. Jakmile dojde k zadání a potvrzení vloženého jména, ovládání je zpřístupněno. Dolní část obrazovky opět obsahuje patičku s hesly, jež jsou po chvíli automaticky ukryta, neboť uživatel je již seznámen s tímto prvkem a není třebas ním nadále narušovat pozornost. Patičku lze případně zobrazit šipkou v původní oblasti prvku a následně zase skrýt.

Obrazovka místnosti je rozdělena na dvě části: levá, menší část patří místnímu uživateli a reprezentuje jeho data a chování aplikace. Pravá část podbarvená růžovou barvou patří ostatním účastníkům místnosti. Ta je v případě neúčasti žádného člena vyplněna polem pro přizvání, jak lze vidět na obrázku 6.4.



Obrázek 6.4: Stránka místnosti obsahující jednoho uživatele s podporou grafického prvku pro přizvání dalších účastníků.

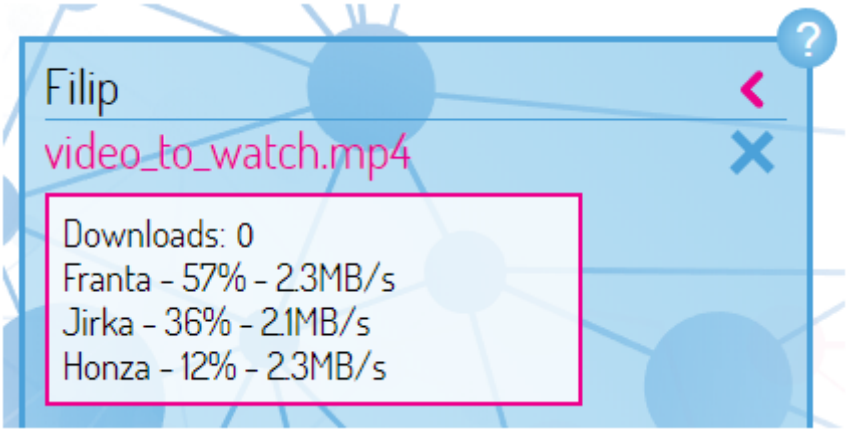
V průběhu plnění místnosti uživateli probíhá přizpůsobování aplikace jejich aktuálnímu počtu, zejména velikosti obdélníků reprezentujících jednotlivé členy. Kvůli omezené velikosti obrazovky nelze zobrazovat libovolný počet těchto obdélníků, vzhledem k zachování minimálního prostoru pro prezentaci vnitřních dat. Proto jsou při maximální velikosti okna aplikace prezentovány maximálně tři entity, další jsou schovány pod vertikálním rolováním, jak lze vidět na obrázku 6.5.



Obrázek 6.5: Obrázek místnosti obsahující čtyři a více uživatelů, kteří si navzájem sdílí své soubory.

Od dvou a více členů v místnosti není zobrazen zmíněný obdélník pro přizvání, jeho funkci supluje nápis `INVITE` v pravé horní části. Opustit místnost lze kliknutím na nápis nesoucí název služby `SPREAD IT OUT` v levé horní části. Zejména tato dvě uvedená tlačítka mohou být těžko odhalitelná, a proto jsem v bublině obdélníku reprezentujícího místního uživatele umístil ikonu otazníku. Po najetí na ni jsou nad všemi funkčními prvky a důležitými oblastmi zobrazeny bubliny, tak zvané *tooltipy*, které upozorňují na prvek a jeho význam. Na samotný otazník je upozorněno při vstupu do místnosti dialogem. Těmito kroky jsem cílil na zvýšení intuitivnosti celé aplikace.

Základem oblasti modrého obdélníku obsahující všechny důležité informace místního uživatele je již zmíněné jméno, jež lze libovolně v průběhu relace měnit. Dále následuje výčet všech vložených souborů. Z důvodu omezení prostoru pro název každého souboru, který se nemusí do tohoto prostoru vejít, dochází ke skrytí zbytku přetékajícího názvu, po najetí je název zobrazen zvětšením výšky vymezeného prostoru. Další informace, které uživatel ocení, jsou zobrazeny po kliknutí na soubor, kdy je zobrazen počet dosavadních stažení a výčet všech probíhajících přenosů, jak lze vidět na obrázku 6.6. Nadlimitní množství souborů je řešeno prostřednictvím rolování v oblasti výčtu. V patičce je uživatel průběžně obeznámen s celkovou rychlostí stahování a nahrávání, které jsem musel řešit iterativním sčítáním jednotlivých datových toků na základě pravidelného volané události, k čemuž jsem využil metodu `setInterval` její pravidelnost je uvedena v tabulce konstant, příloha B. Patička také obsahuje ikonové tlačítko pro přidání souborů, na základě něhož je otevřen dialog, a uživatel vybírá vložení souborů nebo celých složek.



Obrázek 6.6: Výsek z ovládacího prvku, který slouží pro prezentaci místního uživatele, zobrazující soubor s výpisem právě probíhajících přenosů.

Prostor růžový obdélník, reprezentující jiného uživatele je kvůli jednotnosti koncipován velmi podobně. Hlavička obsahuje název uživatele, avšak nezměnitelný, dále následuje výčet souborů s názvem a velikostí. Soubor se může nacházet ve třech stavech,jak lze vidět na obrázku 6.7, a to jako nestažený, stahovaný a stažený. Stahovaný soubor ukazuje uživateli rychlost stahování, množství stažených dat a celkový stav v podobě zbarveného sloupce. V patičce lze nalézt ikonová tlačítka, jimiž lze provést akce označit vše, odznačit vše, stáhnout, přerušit stahování a opětovně stáhnout. Tato tlačítka jsou k dispozici na základě aktuálního výběru, který se provádí kliknutím na jednotlivé soubory.

Obrázek 6.7: Výsek z ovládacího prvku, který slouží pro prezentaci jiného než místního uživatele, zobrazující soubory, kde každý disponující jiným stavem (nestažený, stahovaný, stažený).

6.1.6 Sdílení

Sdílení neboli přizvání účastníků do místnosti je pro aplikaci velmi důležité, a proto mu byla věnována značná pozornost. Aplikace nabízí tři způsoby sdílení, ale principiálně se vždy jedná o sdílení webové adresy, jejímž parametrem je jméno místnosti.

Prvním způsobem je zkopírování adresy do schránky, odkud jej uživatel může distribuovat podle libosti. Pro kopírování do schránky byla použita metoda `document.execCommand`.

Dalšími způsoby sdílení je mířeno na nejpoužívanější sociální síť na světě, a to na Facebook[4]. Prostřednictvím účtu určeného pro vývojáře a knihovny pro napojení na Facebook API[5] byla propojena aplikace se sociální sítí.

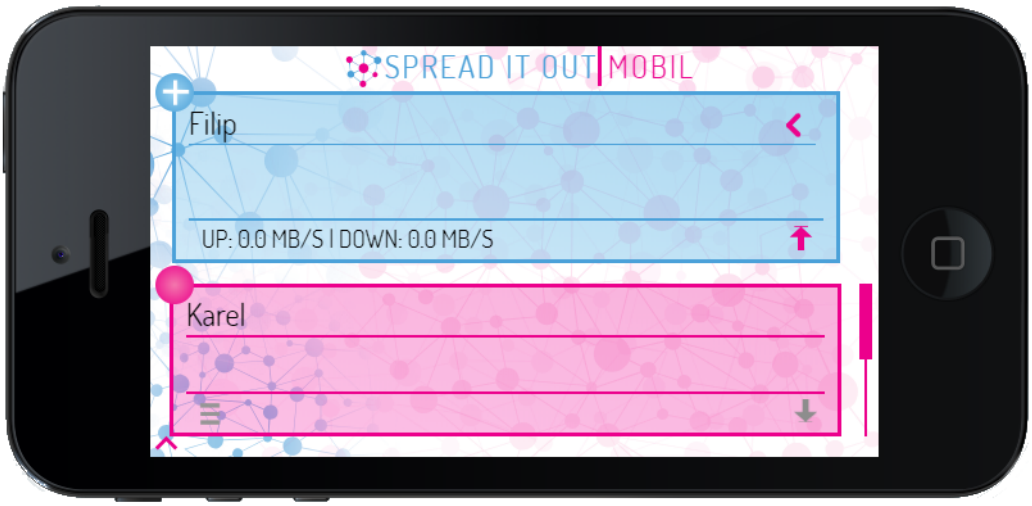
Druhý způsob sdílení se tedy opírá o službu Facebook Messenger, konkrétně lze z aplikace odeslat odkaz na místnost svým přátelům v rámci sociální sítě. Tento vzkaz je doplněný o doprovodnou zprávu.

Třetí způsob vloží status na zeď facebookového profilu, text je fixní a obsahuje odkaz na danou místnost. Pro využití vložení statusu je nezbytné povolit aplikaci zacházení s profilem, k čemuž je uživatel vyzván. Nezbytnou podmínkou pro obě předchozí metody je přihlášení k sociální síti, k němuž je případně uživatel v rámci aplikace vyzván.

6.1.7 Responzivita

V rámci návrhu bylo počítáno s využitím aplikace na tabletech, mobilních a jiných zařízeních, která podporují připojení k internetu a chod webového prohlížeče. Tím aplikace může být zobrazena na velkém množství různých zařízení s variabilně velkými obrazovkami o různých poměrech. Proto je aplikace implementována tak, aby se přizpůsobila jakémukoliv rozměru či poměru. Přizpůsobení lze vidět na obrázcích 6.8 a 6.9.

Responzivita je implementována hlavně za pomoci kaskádových stylů s hlavním využitím prvku `media screen`, díky kterému lze aktivovat a deaktivovat sady kaskádových stylů, a tím plně přeorganizovat uspořádání obrazovky. V případě variabilní velikosti obdélníků reprezentujících uživatele v místnosti jsem použil JavaScript pro úpravu prvků v rámci DOM, a to z důvodu dalšího parametru představujícího celkový počet účastníků, který nelze zohlednit v rámci kaskádových stylů.



Obrázek 6.8: Ukázka responzivity aplikace v případě mobilního telefonu orientovaného na šířku.

[4] https://facebook.com/
[5] https://developers.facebook.com/

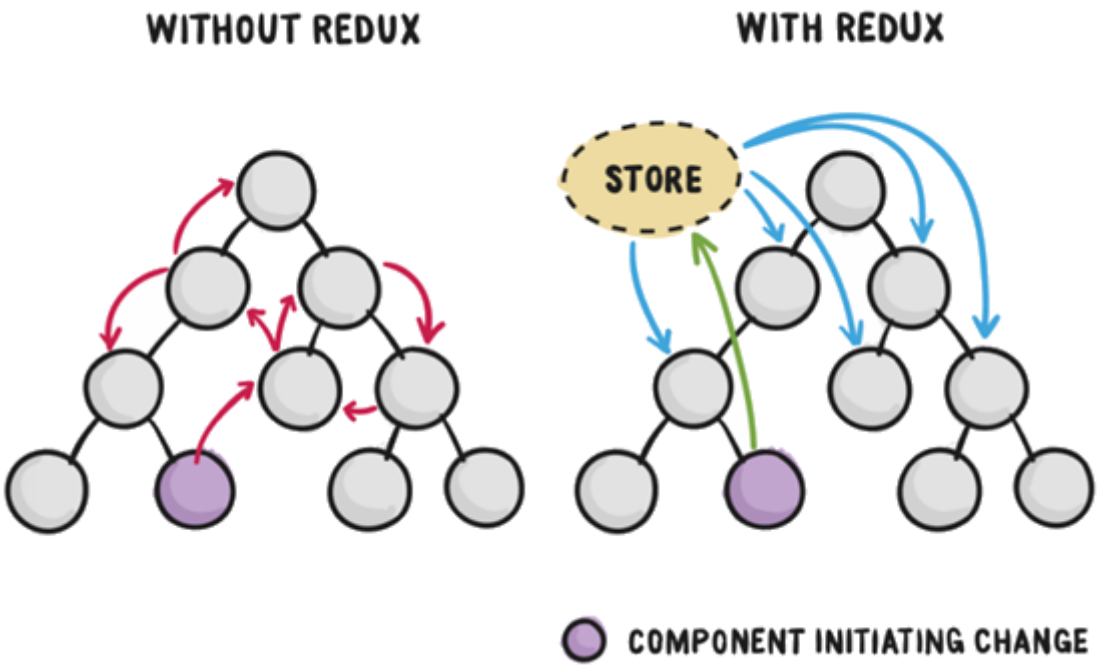
Obrázek 6.9: Ukázka responzivity aplikace v případě mobilního telefonu orientovaného na výšku.

6.1.8 Uspořádání komponent a správa dat

Implementované uživatelské rozhraní je vcelku složité a rozsáhlé vzhledem k velkému množství navzájem propojených komponent. Z tohoto důvodu jsem použil návrhový vzor MVC (model-view-controlller), kde pro komponenty model a kontrolér byla použita knihovna Redux[6] a pro vykreslování pohledů aplikační rámec Polymer.

Knihovna Redux zavádí jednosměrný tok informací, jehož základem je `Store`, který obsahuje celý stav aplikace. Uživatel nebo síť vyvolávají události, jež komponenty registrují a vytvářejí akce, které jsou prostřednictvím metody `dispatch` předány do `Store`, kde dochází skrze `reducer` k aplikování akce na stav, čímž dojde ke změně. Na stav jsou napojeny jednotlivé komponenty, které reagují na tuto změnu, a dojde k překreslení. Toto chování je znázorněno obrázkem 6.10.

[6] https://redux.js.org/



Obrázek 6.10: Levá část obrázku demonstruje tok dat mezi zdrojovou a cílovou komponentou, prostřednictvím komponent mezi nimi.Pravá část obrázku prezentuje tok dat prostřednictvím knihovny Redux, kde stav aplikace je uložen v Store, jehož změna je distribuována všem zainteresovaným komponentám. Zdroj [38].

Definované komponenty jsou vzájemně zanořeny, čímž tvoří stromovou strukturu. Bez uvedené architektury by při vzájemné komunikaci dvou komponent docházelo k protékání dat napříč celým stromem, což by zbytečně zatížilo zprostředkující komponenty a celkově celou aplikaci. Dalším dopadem by bezesporu byla horší udržovatelnost a robustnost ke změnám.

Základní komponenta, která obsahuje celou aplikaci, nese název `sio-app`. Tato komponenta obsahuje jednotlivé stránky a stránky zase obsahují komponenty, z nichž se skládají. Některé komponenty, jako třeba komponenta reprezentující jednotlivé uživatele v místnosti nebo jednotlivé soubory, jsou vykresleny až na základě obdržených dat.

6.1.9 Jazykové překlady

Koncept celé aplikace a jejího ovládání se může novému návštěvníkovi zdát trochu složitý. Sice aplikace disponuje vcelku velkým množstvím různých textů hovořících o principu jejího fungování, ale pokud uživatel neovládá jazyk, v kterém je aplikace přeložena, není mu to moc platné.

To je důvod, proč je aplikace implementována jako vícejazyčná, a to momentálně v jazycích čeština a angličtina. Jednotlivé texty nejsou pevně zakódovány v HTML dokumentech, ale při inicializaci jsou doplňovány JavaScriptem. Pro jednotlivé texty je výchozí dokument ve formátu JSON obsahující jednotlivé dialogy v zadaných jazycích. Doplnění nového jazyka je tak velmi lehké, stačí daný JSON obohatit o nový jazyk a není třeba se dál o nic starat, aplikace si jazyk rozezná a nabídne uživateli k přepnutí. Takto lehce může

do svého jazyka udělat překlad jakýkoliv fanoušek aplikace nebo najatý překladatel, z nichž ani jeden nemusí rozumět žádnému z použitých implementačních jazyků.

Aplikace nabízí přepnutí jazyka v pravém horním rohu prostřednictvím rolovacího menu.

6.2 Zpracování souborů a složek

V následující kapitole se lze dočíst, jakým způsobem jsou zpřístupněny soubory a složky pro sdílení v rámci aplikace, dále taktéž jejich příprava pro samotné odesílání a v neposlední řadě samotný princip pro jejich efektivní odesílání.

6.2.1 Vkládání

Prvním krokem pro sdílení v rámci implementované aplikace je zpřístupnění potřebných souborů a jak bylo řečeno v kapitole 2.2 v rámci webových technologie není možno k souborům přistupovat jinak než vložením souborů uživatelem. Nabízí se zde dvě metody, a to prostřednictvím vstupního pole `<input>` typu `file` nebo Drag and drop API. V rámci implementace jsem použil obě dostupné metody.

Pro první metodu jsou použity dvě vstupní pole. Jedno pro vkládání souborů, s příznakem `multiple` pro vložení více souborů během jedné relace souborového manažera, a druhé vstupní pole s příznaky `webkitdirectory` a `directory` pro vkládání složek. Důvodem této implementace je rozdílnost souborového manažera, kde v rámci souborů lze vybírat pouze soubory a v případě složek jsou zobrazeny pouze složky souborového systému bez souborů. Tyto dva souborové manažery nelze sloučit a proto je uživatel povinen zvolit typ organizační jednotky souborového systému.

Druhá metoda, kterou jsem implementoval, nese název *drag and drop*. Jak název napovídá, jedná se o vložení v podobě přetáhnutí do oblasti okna aplikace. Pro tuto funkcionalitu jsem využil knihovnu drag-drop[7], jež zaobaluje původní API, a tím ulehčuje práci vývojáře pro zavedení této metody.

6.2.2 Příprava složek

Soubory vložené uživatelem jsou zpřístupněny přes File API. V případě vložení celé složky, která zpravidla obsahuje více než jeden soubor, a případně další složky, je k dispozici pole souborů, kde je mimo jiné u každého souboru uvedena cesta v daném adresáři. Z této informace lze vyčíst uspořádání souborů a existenci podsložek v načtené složce.

Problém, jenž jsem v rámci implementace musel vyřešit, spočíval v uložení složky tak, jak byla u odesílatele načtena. Vzhledem k tomu, že není žádným způsobem podporováno uložení složky, bylo by potom zapotřebí soubory z načtené složky ukládat jeden po druhém, a tím by celá hierarchie původní složky zanikla. Z tohoto důvodu byla použita knihovna ZipJs, díky níž aplikace soubory v rámci vložené složky zkomprimuje do jednoho souboru, který je přenesen a uložen. Cílový uživatel tento obdržený soubor rozbalí a získá složku, jež byla předmětem přenosu.

Knihovna ZipJs provádí komprimaci na základě paměti RAM, výstupem je datový typ blob, s nímž se i pojí již zmíněný limit z kapitoly 2.3. Tím je vložená složka limitována na celkové velikosti. Chování aplikace je proto tomuto faktu přizpůsobeno a zavádí pro uživatele velikostní limit. Splnění limitu je ověřováno součtem velikostí všech souborů obsahujících vložená složka.

[7] https://github.com/feross/drag-drop

6.2.3 Odesílání souboru

Odesíláním souborů rozumíme, na základě předchozí kapitoly, i odesílání vložených složek, které jsou v rámci přípravy komprimovány do jednoho souboru formátu ZIP. Jakmile aplikace disponuje soubory, může je začít posílat skrze WebRTC. Odesílání probíhá posupně, a tak je nutné v případě větších velikostí servírovat soubor postupně po kouscích, tak zvaných *chuncích*. Zde se nabídly dva přístupy, oba dva byly implementovány a následně porovnány.

První přístup nahrává celý soubor v podobě jednoho či více blobů do paměti RAM, z nichž se následně provede čtení pro všechny potřebné přenosy. Toto řešení disponuje velkou výhodou v podobě rychlosti vykonávaných operací, a to obzvláště potřebné operace `slice`, kterou blob implementuje. Nevýhodou tohoto řešení je zbytečné zahlcení paměti RAM, v případě vložení velikostně větších souborů, kdy paměť začne být značně zahlcená, v horším případě nedostatečná. Tehdy operační systém započne odkládání dat na pevný disk, což je náročné a zaneprazdňuje disk, procesor a vcelku celou sběrnici. Celé uvedené chování má dopad nejen na aplikaci, ale i na celé zařízení, jež se značně zpomalí.

Druhý přístup získává potřebné kousky souboru postupným řezáním, opět metodou `slice`, kterou implementuje i soubor v rámci File API, což má však jeden háček. Kousky, po nichž se soubor odesílá, jsou vcelku malé, a tak je prováděna spousta žádostí na čtení z pevného disku. Aplikace se dostává do situace, kdy vyřizování žádostí na čtení není tak rychlé jako samotný přenos a celková rychlost přenosu je tak zbytečně limitována. Žádost čtení z pevného disku, krom samotného čtení dat, je časově náročné, vzhledem k nákladům na operaci v podobě (vystavění, ustálení, čtení). Abych tento proces optimalizoval, zkombinoval jsem obě uvedené metody.

Aplikace čte soubor po značně větších kusech, než jsou velikosti odesílaných kousků, a ukládá je do paměti RAM. Tyto kusy dále rozkládá na kousky vhodné pro odeslání. V rámci přenosu jsou pro každý přenos v paměti připraveny dva kusy, první, který je odesílán, a druhý, jenž navazuje na první. Takto je značně zredukován počet přístupů na disk, paměť RAM není zdaleka tak zahlcena a přenos dosahuje značně vyšších rychlostí. Tento fakt byl experimentálně ověřen.

6.3 Přijmutí souboru

Hlavní problémem v rámci implementované aplikace bylo vyřešení komplikace postupného ukládání přijímaného souboru v rámci WebRTC spojení. Toky dat jsou zpracovávány prostřednictvím JavaScriptu, jak již bylo zmíněno, a tedy není možné soubor přijímat běžným způsobem jako `http/s` žádost, s čímž se většinou při běžném stahování setkáváme. Přijímaný datový tok nelze postupně ukládat na disk tak, jak je postupně přijímán, a tedy je zapotřebí stanovit prostor, kam přijímaná data budou ukládána. Jakmile je celý soubor přijat, lze ho jako celek uložit na pevný disk.

Následující kapitola se věnuje dostupným prostředkům pro ukládání přijímaného datového toku, omezením v rámci jednotlivých webových prohlížečů a dalšími řešeními. V závěru kapitoly je popsáno, jaké prostředky byly v rámci implementované aplikace použity.

6.3.1 Použití FileSystem API

FileSystem API, který je implementován v rámci jádra Chromium, se ukázal jako nejlepší řešení pro ukládání přijímaného souboru. Toto API dovoluje aplikaci ukládat libovolně

objemné soubory, bez přílišného zatížení operační paměti RAM a omezení rychlosti zpracování.

Při inicializaci aplikace probíhá zjištění existence metod `window.requestFileSystem` nebo `window.webkitRequestFileSystem`, jež jsou metodami prezentujícími implementaci FileSystem API. V případě existence jedné z uvedených metod je metoda zavolána, čímž aplikace žádá o prostor pro ukládání souborů. Prvním z parametrů volané metody je zadána velikost požadovaného prostoru a druhým typ prostoru. V rámci aplikace jsem zvolil práci s dočasným souborovým prostorem, neboť soubor je vyžadován jen po dobu platnosti relace, čímž zároveň není třeba vyžadovat povolení od uživatele, a třeba i riskovat případné odmítnutí.

Jakmile je prostor přidělen, aplikace může zahájit ukládání přijímaných souborů do vyhrazeného prostoru. V rámci každého přijímaného toku je reprezentujícím přijímaným souborem vytvořen soubor ve vyhrazeném prostoru, do kterého je přijímaný datový tok ukládán.

Po dokončení přenosu webový prohlížeč započne kopírování souboru z vyhrazeného prostoru. Uživateli je toto kopírování prezentováno jako klasické stahování, avšak velmi rychlé, neboť dochází ke kopírování v rámci pevného disku.

6.3.2 Použití StreamSaver.js

Pro jeden z pokusů vyřešení problému s ukládáním přijímaného souboru byla využita i uvedená knihovna. Výhodou této implementace bylo přímé ukládání souboru do stahovaných souborů, a tedy i uživatel mohl reálně vidět průběh stahování mimo aplikaci prostřednictvím manažera stahování. Následně bylo však naraženo na omezení, které je zmíněno v kapitole 2.5, tedy maximální doba přenosu limitována na dobu 330 sekund, a jež v dokumentaci knihovny není uvedeno. Zároveň v době implementace každý webový prohlížeč podporující knihovnu StreamSaver.js podporoval i FileSystem API.

Implementace a přepnutí na tuto knihovnu byla v aplikaci i přesto ponechána z důvodu pozdější případné podpory. Mozilla Firefox začne v dohledné době implicitně podporovat chybějící komponentu pro chod této knihovny a zároveň věřím, že komunita problém s časovým limitem brzy vyřeší. Pokud se tak stane, lze knihovnu použít pro Mozilla Firefox a odstranit tak limit pro přijímaný soubor pro tento webový prohlížeč.

6.3.3 Ukládání do Blobu

Poslední, zaručenou metodou pro ukládání datového toku je ukládání přijatých dat do operační paměti RAM v podobě datového typu blob, jenž bylo využito pro prostředí, která neimplementují Filesystem API. Postupně přijímaná data jsou připojována k blobu vytvořeného při inicializaci přenosu. Takto se data postupně nabalují tak, jak jsou z datového toku přijímána. Průběh stahování lze vidět pouze v aplikaci, neboť soubor je zpřístupněn až poté, co je přenos dokončen a výsledný blob zkopírován z paměti RAM na pevný disk. Kopírovaný soubor nelze navazovat z více blobů, a proto je celková velikost jednoho přijímaného souboru omezená limitem maximální velikosti blobu.

Vzhledem k variabilnosti maximální velikosti blobu, kterou v mnoha případech nelze detekovat, provede aplikace detekci webového prohlížeče a jeho verze v rámci inicializace. Limity jsou pevně stanoveny pro každý webový prohlížeč v tabulce konstant, příloha B. Spojením zjištěného prostředí a tabulky se stanoví limit pro maximální velikost přijímaného souboru. Někdy je limit možno stanovit, a to zvlášť u webových prohlížečů založených na jádru Chromium, kde je limit uveden v proměnné `window.performance.memory.jsHeapSizeLimit`.

Další limit nastává v případě, kdy uživatel začne stahovat více objemných souborů a jejich data se začnou hromadit v paměti RAM. Tímto lze paměť vcelku lehce zahltit i přes to, že operační systém začne včas odkládat na pevný disk. Aplikace proto i v tomto případě disponuje limitem pro maximální velikost přijímaných dat, která je součtem velikosti všech právě přijímaných souborů.

6.3.4 Alternativní řešení

Rozebíraný problém této kapitoly je pro aplikaci zásadní a degraduje celkové použití aplikace. Proto jsem rozmýšlel i o různých alternativních řešeních, která by se dala implementovat a vyřešila daný problém. Jeden z nápadů se opřel o rozdělení souborů na více částí a dát uživateli možnost, aby si je následně sám spojil. V této situaci se nabízí komprimované formáty, jako jsou např. ZIP, RAR, TAR, GZ a tak dále, které lze při komprimaci rozdělit na více částí. Potom by stačila knihovna, jež by dokázala postupně vytvářet jednotlivé části komprimovaného souboru z jednotlivých blobů. Uživatel by ve výsledku stahoval komprimovaný soubor rozdělený na několik části, z nichž by si přijatý soubor rozbalil.

Bohužel v době implementace neexistovala žádná JavaScriptová knihovna splňující kladené nároky, a proto jsem toto řešení neimplementoval.

6.3.5 Výsledné chování

Aplikace je implementována pro využití za nejlepší dostupné metody a případně i limitu pro ukládání přijímaného souboru. Při inicializaci proto postupuje následně uvedeným postupem, dokud není úspěšná. V prvním kroku je proveden test dostupnosti FileSystem API, který je nejlepším dosavadním řešením, a zažádá o přidělení prostoru pro ukládání. V případě neúspěchu zbývá použití metody ukládání do operační paměti RAM, a proto je proveden pokus o stanovení maximální velikosti, a to detekcí prostředí uvedené v kapitole 6.3.3. Pokud ani v tomto kroku není aplikace úspěšná, například v případě použití alternativního prostředí, potom je použita implicitní hodnota maximální velikosti blobu.

Ukládání přijímaných souborů se ukázalo v rámci této diplomové práce jako největší komplikace. Problém jsem detailně nastudoval a snažil jsem se naleznout nejlepší možné řešení pro každý individuální webový prohlížeč. Proto se domnívám, že v době implementace bylo využito všech dostupných prostředků pro docílení nejlepšího možného řešení pro ukládání souborů přes přijímaný datový tok zpracovaný jazykem JavaScript.

6.4 Serverová část a komunikace

Serverová část je rozdělena na samotnou databázi a aktivní prvek. Databáze slouží k uložení veškerých potřebných dat a disponuje prvky pro upozornění na jejich změnu. Aktivním prvkem je myšlen prvek, s nímž každá relace naváže spojení, a tím umožní jejich vzájemné propojení. Dále bude rozebrán samotný peer-to-peer přenos, a to od jeho nastavení, inicializace, směřování, přenosu a následného ukončení.

6.4.1 Parse Server

Základem pro celou aplikaci je prostředek pro ukládání dat s podporou upozornění na změnu, tedy přidání, smazání nebo úpravu dosavadního záznamu. Důvodem k jeho zvolení byl samotný koncept aplikace, která pracuje s místnostmi, jejíž data se průběžně mění a všichni účastnící dané místnosti musí být o změnách informováni. Standardní SQL databáze

by tak vzhledem k chybějící podpoře oznámení o změnách nebyla pro takové ukládání vhodná.

Existují platformy, které tyto požadavky hravě splňují, příkladem může být populární služba Firebase, jež se ale od překročení jistého limitu stává placenou. S ambicemi o vytvoření populární služby bylo zapotřebí použít něco s otevřenou licencí. Dle stanovených požadavků byla vybrána platforma Parse Server založená na NoSQL databázi MongoDB. Data jsou ukládána jako objekty disponující unikátním identifikátorem, na základě kterého se na sebe mohou navzájem odkazovat.

Pro navrženou funkcionalitu bylo zapotřebí ukládat tři typy objektů. První typ reprezentuje místnost, jež je stavebním kamenem pro ostatní data. Tento objekt obsahuje tři atributy, a to název, heslo a unikátní identifikátor v rámci databáze. Druhý typ objektu představuje člena místnosti a mezi jeho atributy patří jméno v rámci místnosti, unikátní identifikátor v rámci databáze, unikátní identifikátor použitého webového prohlížeče, odkaz na objekt místnost a příznak aktivity, jehož hodnota vypovídá o právě aktivním připojení do služby. Poslední, třetí typ objektu reprezentuje soubor o atributech název a velikost souboru, unikátní identifikátor v rámci databáze a odkaz na objekt typu člen místnosti. Všechny objekty databáze, mimo již uvedených, vlastní atributy zaznamenávající jejich poslední úpravu, datum vytvoření a práva pro daný objekt. Atribut s datem úpravy a vytvoření lze do budoucna využít pro provedení údržby nad vytvořenými místnostmi.

Platforma Parse Server automaticky implementuje rozhraní, jehož prostřednictvím lze získat uložené objekty. Tyto dotazy je možné blíže specifikovat podmínkami pro určení potřebné množiny dat. Nad definovanými objekty lze stanovit oprávnění pro operace nad nimi vykonávané z důvodu zabezpečení proti neoprávněnému přístupu. Vzhledem k absenci vedení jakýchkoliv účtů, na které by se práva dala aplikovat, dochází pouze k neautorizovaným přístupům, při nichž lze obtížně definovat, jaká relace by měla mít oprávnění provádět akce nad danými objekty a jaká nikoliv. Z tohoto způsobu ale vyplývají bezpečnostní rizika v podobě přístupu k nežádoucím informacím nebo manipulace s daty. Jeden z příkladů možných nebezpečí může spočívat v rámci přístupu k objektu místnost. Pro relaci musí být zpřístupněno vyhledání objektu místnost, do níž chce uživatel vstoupit, ale zároveň je nežádoucí získání více než jednoho objektu místnost, v horším případě všech objektů typu místnost. Tyto dvě situace nelze v dotazu prostřednictvím práv rozlišit.

Z důvodu výše uvedeného problému jsou všechny dotazy řešené prostřednictvím vlastních definovaných funkcí v rámci Parse Serveru, označovány jako *Cloude Code*. Tyto funkce v rámci volání obdrží parametry a vrací hodnotu, jejímž obsahem je mimo jiné příznak povedeného, či nepovedeného vykonání dané funkce. V případě úspěšného provedení jsou z databáze získána data, která jsou obsahem návratové hodnoty, a v případě neúspěchu dojde v rámci návratové hodnoty k oznámení typu chyby doprovázené chybovou hláškou. V rámci implementované aplikace byly použity dvě funkce, jejichž význam bude dále popsán.

Než bude rozebrána samotná komunikace uživatele s databází, je zapotřebí zmínit způsob a důvody identifikace uživatele v rámci celé aplikace. Prvním důvodem je zamezení vstupu více oken do stejné místnosti a druhým je identifikace uživatele při opětovném vstupu do stejné místnosti z důvodu zapamatování si zadaného jména. Z těchto důvodů bylo zapotřebí najít způsob identifikace uživatele nebo alespoň jeho prostředí, tedy webového prohlížeče. Za tímto účelem byla použita paměť pro uložení dat do paměti webového prohlížeče, do které je uložen vygenerovaný unikátní identifikátor. Uložená hodnota do uvedené paměti se taktéž označuje jako *cookie*. Tento identifikátor je dále používán pro identifikaci uživatele při komunikaci s databází a aktivním prvkem.

První funkce, definovaná jako `join`, je volána při pokusu o vstup do místnosti. Argumenty této funkce jsou jméno místnosti, unikátní id webového prohlížeče a případné heslo. V prvním kroku je vyhledán objekt odpovídající zadanému názvu místnosti. Pokud není nalezen, dojde k vytvoření tohoto objektu typu místnost se zadaným jménem, případně heslem v argumentu funkce. V případě již existující místnosti dojde k ověření hesla. Pokud heslo není správné nebo není v argumentu zadáno, přestože místnost heslo vyžaduje, funkce vrací chybovou odpověď. V případě úspěšného získání a ověření či jenom vytvoření místnosti dochází následně k vyhledání všech objektů typu člen navázaných na tuto místnost. Pokud v získané množině členů neexistuje objekt člen s odpovídajícím unikátním id webového prohlížeče zadaným v argumentu funkce, je vytvořen. Ze získané množiny objektů jsou vyfiltrovány ty, jejichž příznak aktivity je negativní. Ke každému ze zbylých objektů jsou vyhledány všechny objekty typu soubor odkazující se na daný objekt typu člen. Tato funkce zjednodušeně ověří danou místnost a vrátí všechny aktivní členy a jejich soubory, v důsledku čehož dochází k reprezentaci aktuálního stavu a obsahu místnosti.

Druhá funkce, definovaná jako `isFree`, slouží k ověření existence zadané místnosti. Funkce obdrží název místnosti a následně se v databázi pokusí vyhledat místnost s tímto jménem, výsledek je v podobě booleovské proměnné `isSet` vrácen volající relaci.

Jakmile dojde k získání dat místnosti, následuje napojení na události pro objekty typu člen a soubor spojené s místností prostřednictvím tak zvaných *LiveQueries*, která jsou definována tak, aby uživatel obdržel události zmíněných objektů vztahující se pouze k zadané místnosti. Události, s nimiž je relace obeznámena, jsou typu vytvoření, smazání a modifikace objektu daného typu. Takto se udržují všechny relace aktuální, neboť při změně dat v databázi vztahující se k určené místnosti jsou ihned obeznámeni, a tak vlastní vždy aktuální stav databáze.

Další operace pojící se k databázi jsou vytvoření souboru, smazání souboru a modifikace člena místnosti. Tyto operace lze provádět pouze se znalostí daných objektů, respektive jejich unikátních identifikátorů, jež bez připojení do místnosti a voláním funkce `join` nelze získat. V případě vytvoření objektu typu soubor je zapotřebí znát referenční objekt typu člen.

6.4.2 SocketIo

Dalším prvkem celé topologie je knihovna SocketIo, která plní dvě důležité funkce. První, nezbytná funkcionalita je detekce připojení nebo odpojení relace. Druhou je pak provedení signalizace pro ustanovení peer-to-peer přenosu.

V rámci vstupu do místnosti je prostřednictvím funkce `join` vykonané na straně Parse Serveru vytvořen objekt typu místnost a objekt typu člen, jehož aktivita je určena příznakem `isActive`, který se při připojení a odpojení musí změnit. Připojení do místnosti není v podstatě komplikace, neboť v rámci vykonávání funkce `join` je zřejmé, že člen je aktivní, a tedy lze jeho příznak aktivity nastavit na pozitivní. Problém nastane při jeho odpojení, což prostřednictvím Parse Serveru nelze detekovat. Tuto funkcionalitu právě zastupuje SocketIo server, jenž je schopen detekovat uzavření spojení.

Vstup do místnosti po obdržení všech dat od Parse Serveru pokračuje v ustanovení spojení se serverem SocketIo. V rámci ustanovení tohoto spojení zadává vstupující člen id místnosti, do níž vstupuje, heslo, pokud je místnost pod heslem, a unikátní identifikátor webového prohlížeče. Server ověří existenci místnosti, případně i heslo, a příznak aktivity objektu člena s odpovídajícím identifikátorem nastaví jako aktivní. Jakmile je příznak změněn prostřednictvím Live Queries, obdrží všichni již připojení členové informaci o nově

připojeném členovi. Vytvořené spojení je uloženo do paměti. Uložením do paměti je myšleno uložení do proměnné hodnoty v podobě matice, ve které jsou uspořádána a uložena všechna aktivní spojení. Matice je dvourozměrné pole, v jehož prvním rozměru jsou indexovány aktivní místnosti na základě identifikátoru místnosti a v druhém rozměru definuje všechna spojení v rámci místnosti. Aktivní místností je myšlena místnost o jednom a více aktivním členu. S každým členem jakékoliv místnosti je ustanoveno spojení a dokud je toto spojení aktivní, je i příznak `isActive` patřící objektu, který v databázi reprezentuje daného člena, nastaven na aktivní. Jakmile je spojení uzavřeno, dojde i ke změně příznaku v databázi, o což se postará SocketIo server. Vytváření zmíněné matice pro ukládání spojení má dva významy.

Prvním významem je rychlé indexování potřebného spojení. Při signalizaci funguje server SocketIo k přeposílání signalizačních dat. Zdroj zadá identifikátor cíle a díky této matici se rychle vyhledá vytvořené spojení, jemuž jsou data přeposlána. V druhém případě se jedná o bezpečnostní účel, kdy si díky této matici lze rychle ověřit, že se jedná o komunikaci v rámci místnosti a není zapotřebí při každé zprávě zasílat autorizační data, která by prostřednictvím databáze byla validována.

V případě pokusu o připojení druhého okna v rámci stejného prohlížeče dojde ke kolizi na straně SocketIo serveru v podobě již otevřeného spojení pod daným indexem, což je zjištěno již v samotném zápisu do matice. V tomto případě je připojující se strana s nastalou situací obeznámena zprávou typu `ALREADY_ACTIVE`. Na základě ní obdrží uživatel dialog o již připojené relaci a v případě, že se rozhodne převzít kontrolu, dojde k zaslání zprávy typu `KICK_USER`. SocketIo server zašle vyhozené straně zprávu typu `KICKED`, ukončí s touto stranou spojení, přistoupí do databáze, kde smaže všechny soubory daného člena, a nakonec do matice zapíše nové spojení, kterým obeznámí o zdárném dokončení akce zprávou typu `USER_KICKED`.

Závěrem je zapotřebí dodat, že SocketIo server pracuje nad databází s vyšším oprávněním, a tím jsou mu dovoleny akce, které běžné neautorizované relaci nejsou dovoleny.

6.4.3 Peer-to-peer přenos

Samotný peer-to-peer přenos je mezi dvěma relacemi uskutečněn prostřednictvím knihovny WebRTC, jejíž přímá podoba nebyla použita, namísto toho byla použita knihovna Simple-peer[8], jež je nadstavbou nad WebRTC. Hlavní výhoda této nástavby spočívá v jednodušší a stručnější implementaci ve stylu standardu Node.js, s nímž se mi lépe pracovalo.

Přenos dat je zprostředkován prostřednictvím dvou objektů třídy `Peer`, jednoho na straně odesílatele a druhého na straně příjemce. Vytvářený objekt je nastaven na základě stanovené konfigurace předávané jako parametr při vytváření objektu. Hlavní položkou jsou definice STUN a TURN serverů, které se k propojení použijí, a boolovský příznak `initator` pro rozlišení objektu inicializujícího přenos.

Samotný proces přenosu započne, jakmile uživatel vyvolá akci stáhnout soubor. V tuto chvíli dojde na straně relace, která soubor požaduje, k vytvoření nového objektu třídy `Peer`, jenž slouží ke zprostředkování přenosu. Následuje definice metod volaných při přijmutí dat, uzavření a otevření kanálu nebo výskytu nečekané chyby. Jakmile je vše připraveno k přijímání, jsou vygenerována signalizační data, jež je zapotřebí předat relaci vlastnící požadovaný soubor.

Předání lze přirovnat k poštovní obálce. Pro odesílající signalizační data je vytvořen nový objekt, v rámci něhož je uložen identifikátor odesílatele, identifikátor příjemce, iden-

[8] https://github.com/feross/simple-peer

tifikátor požadovaného souboru, signalizační data a typ zprávy, který je v tomto případě `FILE_REQUEST_SEND`. Následně dojde k odeslání tohoto objektu prostřednictvím otevřeného kanálu se serverem SocketIo. Server data validuje a vyhledá otevřený kanál s cílovou relací, před samotným přeposláním dochází ke změně typu zprávy na `FILE_REQUEST_RECEIVE`.

Cílová strana obdrží tuto zprávu a opět vytváří objekt třídy `Peer`. U tohoto objektu definuje následující metody: otevření, zavření kanálu a výskyt neočekávané chyby. Jakmile je vše potřebné nadefinováno, dojde k předání signalizačních dat. Jakmile jsou signalizační data předána, objekt vygeneruje svá signalizační data, která se musí předat zpět iniciátorovi. Způsob adresace je stejný jako v předchozím odstavci, pouze typ zprávy je stanoven na `FILE_CONFIRM_SEND`. SocketIo server data validuje, následně vyhledá otevřené spojení s cílovou relací a předává data ve zprávě typu `FILE_CONFIRM_RECEIVE`.

Po přijetí zprávy jsou signalizační data předána zpět do zahajujícího objektu typu `Peer`. Následné ustanovení spojení proběhne automaticky prostřednictvím STUN, případně TURN serverů. Jakmile dojde k ustanovení spojení, obě strany tuto událost zaznamenají a zahájí přenos souboru. Přenášení probíhá po kouskách o volitelné velikosti, konkrétní použitá velikost v rámci implementace je uvedena v tabulce konstant v příloze B. Rozkládání souboru na straně odesílatele a následné skládání na straně příjemce je popsáno v kapitolách 6.2.3, 6.3.

6.5 Další vývoj

V rámci této diplomové práce byla vytvořena funkční webová aplikace, která podle mého názoru plně splňuje základní a klíčovou myšlenku peer-to-peer sdílení souborů ve skupině, a to s jednoduchým a originálním vizuálním stylem. I když se aplikace může zdát jako hotovou a ukončenou záležitostí, je tomu právě naopak. V následujících kapitolách uvedu několik směrů, kam by se dal vývoj aplikace posunout.

6.5.1 Nové technologie

Hlavním úskalím, na které aplikace trpí, je již několikrát zmíněný velikostní limit přijímaného souboru v rámci některých webových prohlížečů. Limit značně ubírá na dokonalosti, avšak dřív nebo později lze očekávat zveřejnění nějaké technologie, jež by daný problém mohla vyřešit nebo alespoň navýšit dosavadní limit. Jedním s příslibů je knihovna StreamSaver.js, o jejímž možném přínosu bylo psáno v kapitole 6.3.2.

Z výše uvedeného vyplývá, že je nadále potřeba sledovat dění okolo této problematiky a vyčkávat na možné řešení. Pokud by však bylo žádoucí problém nějakým způsobem vyřešit ihned, potom lze začít implementovat knihovnu, která by dokázala vytvořit komprimovaný soubor rozdělený na několik části, jak bylo řečeno v kapitole 6.3.4. I přestože se jedná o jasné a s velkou pravděpodobností proveditelné řešení, myslím, že pro uživatele nebude moc atraktivní oproti přepnutí do webového prohlížeče založeného na jádru Chromium.

6.5.2 Export z webového prostředí

Dalším možným rozšířením by mohlo být převedení webové aplikace do desktopové, či mobilní. Otázkou je, jaký by mělo smysl dělat uvedený krok, když unikátnost implementované aplikace je právě v přístupnosti v rámci webové stránky bez instalace a jakýchkoliv prerekvizit.

V případě, že pro věrného a stálého uživatele nebude instalace do jeho zařízení překážkou, potom by díky nainstalované verzi získal výhody v podobě lepší optimalizace výkonu a zachování vložených souborů, neboť by aplikace měla přístup k souborovému systému, a v neposlední řadě žádná omezení v podobě limitů pro přijímaný či odesílaný soubor.

Vzhledem k existenci mnoha nástrojů pro převedení webové aplikace do mobilní by samotné převedení nebylo vůbec náročné. Příkladem může být nástroj Cordova CLI[9] pro převedení webové aplikace napříč mobilními platformami nebo nástroje Elektron[10] pro převedení do desktopové aplikace.

[9] https://cordova.apache.org/
[10] https://electronjs.org/

Kapitola 7

Testování

Poslední, neméně důležitou částí této diplomové práce, je testování výsledné webové aplikace, jenž odhalilo chyby a ověřilo kvalitu a správnost implementace. První fáze testování, takzvané integrační testování, byla provedena samotným autorem aplikace a mělo za úkol ověřit správnou funkcionalitu napříč spektrem zařízení, operačních systémů a webových prohlížečů.

Druhá fáze testování byla zaměřena čistě na výkonnost, která taktéž částečně vypovídá o kvalitě implementace. Spolu s implementovanou aplikací byly podrobeny testu i obdobné služby a jiné způsoby přenosu dat mezi dvěma zařízeními. Cílem tohoto testu bylo ukázat kvalitu, unikátnost a výhodnost použití implementované aplikace oproti aplikacím jiným, případně výhodnost oproti jiným způsobům přenosu dat, které jsou v dnešní době nejčastěji využívány.

Další testování bylo ryze zaměřeno na uživatelské prostředí. První fáze tohoto testování proběhla na úzké skupině uživatelů. Jejich poznatky a způsob ovládání aplikace se staly předmětem k její úpravě, cílem úprav bylo zlepšit ovládání a intuitivnost.

V poslední fázi celého testování došlo k volnému rozšíření mezi užší komunitu lidí s podporou zanechání zpětné vazby. Cílem této fáze bylo vyladění posledních skrytých chyb a další uzpůsobení uživatelského rozhraní.

7.1 Integrační testování

Integrační testování, přestože je uvedeno jako první, nelze považovat za zahajující. V průběhu implementace byly prováděny průběžné testy vývojářem pro ověření správnosti implementace právě vyvíjejících se prvků, případně prvků spolupracujících.

Jakmile byla implementace ukončena, započalo integrační testování, jehož hlavním aktérem by měl být nezávislý testovací tým, jehož ustanovení je ale náročné, a proto integrační testování provedl samotný vývojář. Hlavními předměty testování byly responzivita aplikace, funkčnost jednotlivých prvků a samotná podpora i funkčnost peer-to-peer přenosu prostřednictvím knihovny WebRTC, s níž souviselo i ověření nastavených limitů pro maximální velikost datového typu blob. S prostředími, která hlásila chybějící podporu knihovny WebRTC, byl nedostatek prověřen.

Cílem bylo otestovat především výše uvedené předměty testování v co největší množině různých prostředí. Prostředím je myšlena kombinace zařízení, operačního systému a webového prohlížeče. V sekci mobilních zařízení, jejichž prostředí se zdálo nejvíce kritické,

Značka	Model
Apple	Iphone SE
Huawei	P10 Lite
Xiaomi	Mi Note 2
Xiaomi	Redmi 4X
Apple	Ipad Air 16GB

Tabulka 7.1: Výčet mobilních zařízení na nichž aplikace byla otestována.

byl v rané fázi použit emulátor GenyMotion[1]. Díky tomuto emulátoru bylo možné vyzkoušet aplikaci napříč spektrem různých mobilních zařízení s operačním systémem Android o různých verzích. Společnost Apple, která vlastní a vyvíjí operační systém iOS, nedovoluje spuštění operačního systému jinde než na zařízeních této společnosti, což je důvod, proč pro ně nebylo možné emulovat chod aplikace.

Jakmile došlo k vyladění chyb, jež se prostřednictvím emulovaných zařízení vyskytly, pokračovalo testování na fyzických mobilních zařízeních, jejichž výčet je prezentován tabulkou 7.1. Dále byly prověřeny operační systémy Windows, Linux Fedora, Linux Ubuntu, macOS a jimi dostupné webové prohlížeče Mozilla Firefox, Google Chrome, Safari a Opera.

7.2 Výkonnostní testování

Dalším předmětem testování byla výkonnostní stránka implementované aplikace, která je zároveň hlavním předmětem celé diplomové práce. Celý koncept je vyvíjen za účelem jednoduchého a rychlého sdílení souborů prostřednictvím peer-to-peer síťové komunikace, a proto jako vypovídající hodnota rychlosti aplikace bylo zvoleno měření rychlosti přenosu souboru z jednoho zařízení na druhé.

Rychlost přenosu je plně závislá na prostředí, v němž se přenos uskutečňuje, a proto bylo vytvořeno téměř ideální prostředí, které je však bez problému dosažitelné i pro běžného uživatele. V takto vytvořeném prostředí proběhlo testování i s obdobnými koncepty a jinými způsoby přenosu souborů mezi dvěma zařízeními. Prostředí, jež bylo vytvořeno, se skládalo ze dvou stolních počítačů připojených navzájem do síťového přepínače, jenž byl prostřednictvím směřovače připojen do celosvětové sítě. Konfigurace těchto dvou stolních počítačů je znázorněna v tabulce 7.2.

Komponenta	Typ
Paměť RAM	Corsair Vengeance 16GB DDR4 3000
Procesor	AMD Ryzen 5 1600
Základní deska	MSI B350M MORTAR
Pevný disk	Crucial MX300
Síťové propojení	Switch Netgear Prosafe 1000 Mbps

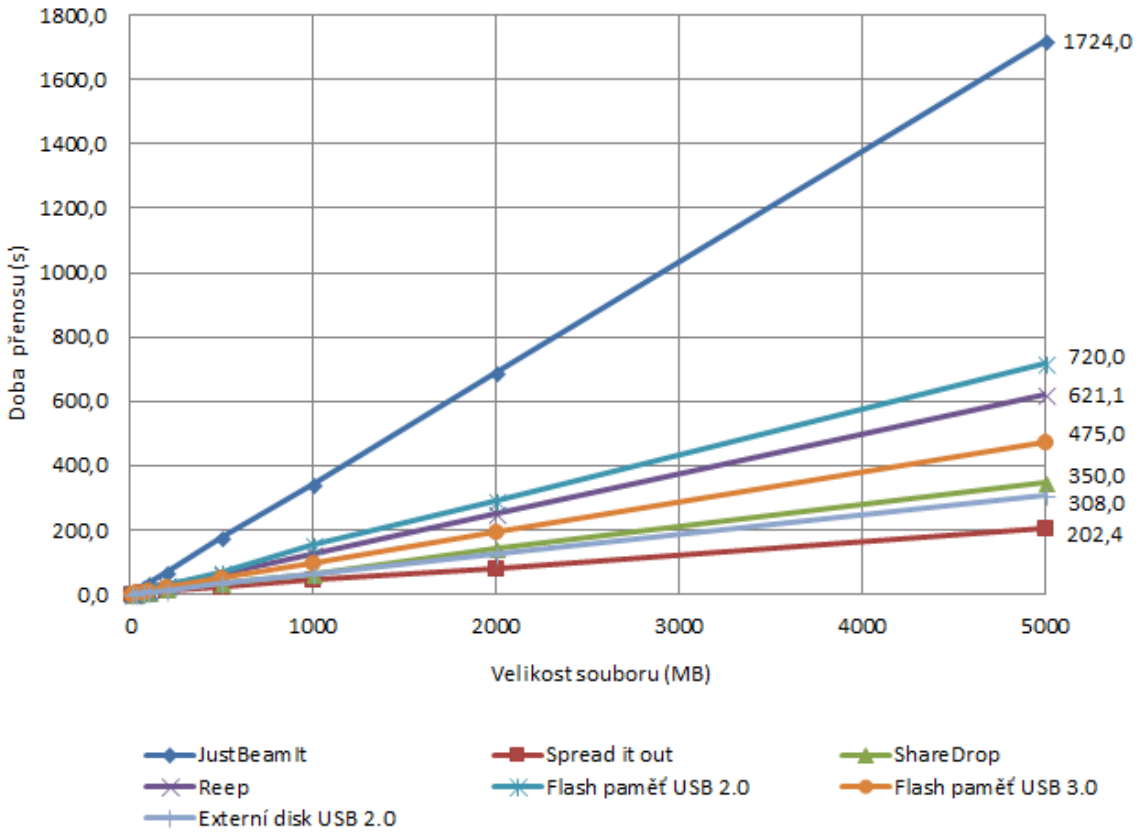
Tabulka 7.2: Výčet komponent, z kterých se skládala zařízení pro testování rychlosti přenosu.

Došlo k otestování a porovnání tří webových aplikací podobného konceptu, jimiž jsou JustBeamIt, ShareDrop a Reep[2]. Dále byly otestovány a porovnány běžně používané pro-

[1] https://www.genymotion.com/
[2] https://reep.io/

středky pro přenos souborů, mezi něž patří flash paměť s připojením USB 2.0, flash paměť s připojením USB 3.0 a externí disk s rychlostí otáčení ploten 5400 rpm (rounds per minute) s připojením USB 2.0. Předmětem kopírování se stalo osm souborů o různé velikosti, které byly postupně přenášeny. Doba přenosu u jednotlivých souborů byla změřena a zanesena do tabulky v příloze A. V případě přenosu v rámci flash pamětí a externího disku došlo k zanedbání doby přemístění nosiče z jednoho zařízení do druhého, a tím je celková doba přenosu rovna součtu dob nahrávání ze zdrojového zařízení a stahování do zařízení cílového.

Výsledky testování jsou prezentovány grafem na obrázku 7.1, z něhož lze usoudit to, že implementovaná aplikace je nejrychlejším prostředkem pro přenos souborů mezi všemi testovanými prostředky. Aplikace přenášela soubory průměrnou rychlostí 23,9 MB/s, což je o 7,9 MB/s větší rychlost přenosu než prostřednictvím externího disku s připojením USB 2.0, který se umístil na druhém místě v rychlosti způsobu přenosu dat. V případě porovnání s konkurenčními službami založenými taktéž na knihovně WebRTC, jako jsou např. ShareDrop a Reep, dosáhla implementovaná aplikace značně vyšších rychlostí, což je nejspíše způsobeno algoritmem zpracování souboru, způsobem manipulace s datovým tokem a nastavením knihovny WebRTC. Dalším prvkem, jenž nejspíše přispěl k výsledné rychlosti, byla optimalizace implementovaných metod podporující zpracování na více vláknech procesoru.



Obrázek 7.1: Graf porovnává rychlost přenášení souborů prostřednictvím vybraných aplikací a prostředků pro přenos souborů mezi dvěma zařízeními.

V průběhu testování bylo taktéž sledováno vytížení prostředků obou zařízení, které se ale zdaleka nepřibližovalo k plnému vytížení. Z tohoto poznatku vyplynula otázka, kde tedy vzniká omezení, vzhledem k mnohem vyšší teoretické rychlosti. Z tohoto důvodu bylo provedeno ještě jedno testování nad implementovanou aplikací, ale v rozdílných operačních systémech. Testovanými operačními systémy se staly Windows, na němž byla provedena i předchozí měření, a operační systém Fedora. Oba tyto operační systémy použily webový prohlížeč Google Chrome. Výsledek testování byl překvapující, jak lze vidět v tabulce 7.3. Prostřednictvím Fedory bylo dosaženo značně vyšších rychlostí než prostřednictvím Windows. Důvodem je nejspíš množství přidělených prostředků, které je v rámci každého operačního systému rozdílné.

Prostředí	Délka přenosu (s)			Průměrná rychlost (MB/s)
	1000 MB	2000 MB	5000 MB	
Windows - G. Chrome	44,4	80,4	202,4	24,5
Fedora - G. Chrome	26,6	52,4	127,8	38,7

Tabulka 7.3: Tabulka porovnávající přenos souborů prostřednictvím implementované aplikace na rozdílných operačních systémech

7.3 Akceptační testování

Další etapa testování probíhala s běžnými uživateli, kteří byli bez teoretické přípravy posazeni před aplikaci, načež došlo ke sledování jejich chování. Testování proběhlo na čtyřech subjektech různé věkové kategorie disponujících průměrnými dovednostmi ovládání a orientování se ve webovém prostředí.

Hlavním nedostatkem při pozorování prvního subjektu bylo samotné zorientování se v aplikaci, ať již po stránce ovládacích prvků, tak i po stránce pochycení podstaty aplikace. Na základě tohoto poznatku byl implementován průvodce aplikací, který je možno spustit ikonou otazníku v úvodní stránce. Průvodce představí hlavní myšlenku aplikace a všechny ovládací prvky v rámci úvodní stránky.

Další, v pořadí druhý, subjekt byl již seznámen s verzí podporující zmíněného průvodce. Nicméně rovněž došlo k zadrhnutí na stejném problému jako u prvního subjektu z důvodu nedostatečné nápadnosti a intuitivnosti implementovaného tlačítka s ikonou otazníku pro spuštění průvodce. Z této skutečnosti vyplynula potřeba ještě lépe nového uživatele navést v případě, kdy si neví rady. Z tohoto důvodu byla implementována detekce prvního přístupu na webovou stránku pomocí *cookie*. Relace, jež se na webové stránce vyskytuje poprvé, je obohacena o dialog upozorňující na funkcionalitu tlačítka s ikonou otazníku.

Třetímu subjektu se zadařilo obstojně zorientovat, avšak postrádal prostředek vysvětlující podstatu aplikace již na první pohled. Tento poznatek byl podnětem pro vytvoření úvodní animace, jejímž obsahem jsou právě základní myšlenky, které mají za úkol uživatele obeznámit s výhodami aplikace a navnadit jej na její používání.

Dalším poznatkem, který všichni uživatelé sdíleli, bylo těžké zorientování se v ovládacích prvcích na stránce místnosti. Z tohoto důvodu byla opět implementována ikona otazníku v oblasti modrého okna reprezentujícího místního uživatele, která při najetí na ni upozorní na všechny ovládací prvky aplikace.

7.4 Testování širokou veřejností

Závěrem celého testování došlo ke zveřejnění implementované aplikace mezi širokou veřejnost prostřednictvím sociální sítě Facebook. Ze začátku tak bylo učiněno v menším měřítku, a to pouze veřejným statusem na zdi autora diplomové práce. Za tímto účelem testování byl vytvořen dotazník, prostřednictvím něhož mohli uživatelé vyjádřit svůj názor. Druhým prostředkem, kterým uživatelé mohli vyjádřit svůj názor nebo nahlásit chybu, byl formulář pod tlačítkem s nadpisem feedback, přímo v aplikaci. Jeho užitím dojde k zobrazení textového formuláře, jehož odesláním je zaslán email autorovi.

Prostřednictvím těchto nástrojů a komunikačních kanálů se uživatelům podařilo najít a nahlásit pár nedostatků, jež byly ihned opraveny. Mimo nahlášených chyb byly taktéž obdrženy užitečné postřehy, zvlášť pro vylepšení ovládání a intuitivnosti aplikace, které byly z části implementovány, zbytek lze využít jako podklad pro další rozvoj. Samotný formulář byl vyplněn čtyřmi uživateli, a tedy z něho nelze vyvodit moc velké závěry. Nicméně jedna z otázek směřovala na úspěšnost přenesení souborů, které se zdařilo všem respondentům. Respondenti taktéž velmi kladně odpovídali na otázku, jestli porozuměli konceptu. Ovládání a intuitivnost aplikace byla v průměru hodnocena mírně nadprůměrně.

Kapitola 8

Závěr

Na začátku této diplomové práce se zrodila vize vytvořit unikátní webovou aplikaci pro peer-to-peer sdílení souborů ve skupině. Cílem se stalo vytvoření aplikace, jež nabídne skupině uživatelů prostor pro sdílení souborů mezi sebou navzájem s podporou takového přenosu, který nebude nijak omezený a nebude vyžadovat prostředků třetích stran.

První krok pro realizaci stanoveného cíle směřoval k zmapování technologií, jež by tento koncept byly schopné utvořit. Mimo technologie nezbytné pro vznik potřebné webové aplikace byla objevena i knihovna pro uskutečnění požadovaného peer-to-peer přenosu, a tím se stal navržený koncept uskutečnitelným.

Další postup směřoval k zmapování již existujících řešení. Jak se ukázalo, podobné aplikace již existovaly, ale žádná z nich nevystihla původní myšlenku vytvoření jakési uzavřené skupiny, v níž si vybraní uživatelé navzájem sdílí soubory všemi potřebnými směry. Dalším nedostatkem, jímž již existující řešení trpěla, byly limity pro maximální velikost přenášeného souboru, které si nově vzniklá aplikace stanovila za cíl překonat.

Výsledkem implementace se stala uživatelsky atraktivní a intuitivní aplikace nabízející uživatelům prostor v podobě místností, v nichž se shlukují uživatelé, vkládají do nich své soubory a navzájem je sdílí. Přístup do těchto místností je stanoven na základě názvu místnosti a případně hesla pro maximální soukromí. Celou místnost lze jednoduše sdílet prostřednictvím URL nebo napojením na sociální síť Facebook. Aplikace synchronizuje stav všech relací, díky tomu tak všichni uživatelé disponují aktuálním stavem místnosti. Další výhodou aplikace je sdílení celých složek.

V průběhu implementace byla objevena komplikace se zpracováním souboru u přijímající strany, která postrádala optimální způsob ukládání přijímaného datového toku. Tento problém byl důkladně analyzován. Výsledek analýzy spočíval ve zjištění chybějícího univerzálního řešení napříč všemi webovými prohlížeči. Nastudováním problému, souvisejících principů a dostupných technologií byl implementován nejlepší možný způsob řešení pro každý webový prohlížeč zvlášť, ale i přesto některé z nich disponují limitem maximální velikosti přenášeného souboru.

Vytvořená aplikace byla spolu s dalšími obdobnými aplikacemi podrobena výkonnostním testům, z nichž vyšla jako nejrychlejší. Porovnání bylo provedeno i s běžně užívanými prostředky pro přenos souborů, taktéž se mezi nimi ukázala jako nejlepší možný prostředek pro přenos souborů mezi dvěma zařízeními.

Testování bylo prováděno taktéž s uživateli. V první fázi pouze s malou skupinou, jejíž chování bylo analyzováno, a na základě získaných podnětů došlo ke stanovení několika úprav. V druhé fázi následovalo zveřejnění prostřednictvím sociální sítě Facebook, díky

které se podařilo získat další zpětnou vazbu, jež dále pomohla aplikaci ještě více vylepšit a vyladit pro veřejné užití.

Výsledkem této diplomové práce se stala aplikace, jejíž koncept, výkonnost a provedení je unikátní a připraveno k volnému rozšíření. Celá aplikace je dostupná pod URL www.spreaditout.eu.

Literatura

[1] *About the Polymer Project*. [Online; navštíveno 27.12.2017].
URL https://www.polymer-project.org/about

[2] *DB-Engines Ranking*. [Online; navštíveno 29.12.2017].
URL https://db-engines.com/en/ranking

[3] *Front-end JavaScript frameworks*. [Online; navštíveno 27.12.2017].
URL https://github.com/collections/front-end-javascript-frameworks

[4] *Just beamIt file transfer made easy*. [Online; navštíveno 31.12.2017].
URL https://justbeamit.com/help

[5] *Our Core Values*. [Online; navštíveno 29.12.2017].
URL https://www.mongodb.com/company

[6] *Peer-to-peer*. [Online; navštíveno 29.12.2017].
URL https://it-slovnik.cz/pojem/peer-to-peer

[7] *Polymer (library)*. [Online; navštíveno 27.12.2017].
URL https://en.wikipedia.org/wiki/Polymer_(library)

[8] *Save Failure - Network Error (due to Service Worker Restart)*. [Online; navštíveno 4.1.2018].
URL https://github.com/jimmywarting/StreamSaver.js/issues/39

[9] *SITES THAT USES OR DEMO WebRTC*. [Online; navštíveno 30.12.2017].
URL http://www.webrtcworld.com/webrtc-list.aspx

[10] *WebRTC Peer-to-peer connections*. [Online; navštíveno 30.12.2017].
URL https://caniuse.com/#search=WebRTC

[11] *Web sockets*. [Online; navštíveno 28.12.2017].
URL https://html.spec.whatwg.org/multipage/web-sockets.html#network

[12] *What is MongoDB?* [Online; navštíveno 29.12.2017].
URL https://www.mongodb.com/mongodb-architecture

[13] *Apple has officially announced support for WebRTC*. Červen 2017, [Online; navštíveno 30.12.2017].
URL https://webrtc.ventures/2017/06/webrtc-support-in-safari-11

[14] Arrachequesne, D.: *socket.io*. [Online; navštíveno 28.12.2017].
URL https://github.com/socketio/socket.io

[15] Bidelman, E.: *HTML Imports*. Listopad 2013, [Online; navštíveno 27.12.2017].
URL https://www.html5rocks.com/en/tutorials/webcomponents/imports/

[16] Bidelman, E.; Loukides, M.; Blanchette, M.: *Using the HTML5 Filesystem API*. United States: O'Reilly, 2011, iSBN13:9781449309459.

[17] Developers, G.: *WebRTC Plugin-free realtime communication*. [Online; navštíveno 30.12.2017].
URL http://io13webrtc:appspot:com

[18] Dutton, S.: *WebRTC in the real world: STUN, TURN and signaling*. Listopad 2013, [Online; navštíveno 30.12.2017].
URL https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/

[19] GRANT, A.: *Beginning AngularJS*. Germany: Apress, 2014, iSBN 1484201612.

[20] Hancke, P.: *WebRTC code samples*. [Online; navštíveno 30.12.2017].
URL https://github.com/webrtc/samples

[21] Hawke, S.: *REST*. Prosinec 2011, [Online; navštíveno 28.12.2017].
URL https://www.w3.org/2001/sw/wiki/REST

[22] HAYWARD; Jonathan; FEDOSEJEV, A.; aj.: *React*. Birmingham : Packt Publishing, 2016, iSBN 9781786462848.

[23] Keefer, C.: *The What and Why of Javascript Framework*. Květen 2015, [Online; navštíveno 27.12.2017].
URL https://artandlogic.com/2015/05/the-what-and-why-of-javascript-frameworks/

[24] Kern, A.; Bald, N.: *Fille Pizza*. [Online; navštíveno 31.12.2017].
URL https://github.com/kern/filepizza

[25] van Kesteren, A.; McCathieNevile, C.; Song, J.: *Progress Events*. Únor 2014, [Online; navštíveno 28.12.2017].
URL https://www.w3.org/TR/progress-events/#Progress

[26] Kincl, P.: *Peer-to-peer sítě*. Listopad 2016, [Online; navštíveno 29.12.2017].
URL https://www.pravniprostor.cz/clanky/ostatni-pravo/peer-to-peer-site

[27] Kodera, J.: *Návrh databáze – NoSQL vs SQL*. Březen 2010, [Online; navštíveno 29.12.2017].
URL https://www.zdrojak.cz/clanky/navrh-databaze-nosql-vs-sql/

[28] Kruisselbrink, M.; Shmyroff, V.: *File API*. Říjen 2017, [Online; navštíveno 28.12.2017].
URL https://www.w3.org/TR/file-upload/

[29] Lardinois, F.; Constine, J.: *Facebook Shutters Its Parse Developer Platform*. Leden 2016, [Online; navštíveno 29.12.2017].
URL https://techcrunch.com/2016/01/28/facebook-shutters-its-parse-developer-platform/

[30] a M. Holdrege, P. S.: *IP Network Address Translator (NAT) Terminology and Considerations*. Lucent Technologies, Srpen 1999, [Online; navštíveno 29.12.2017].
URL https://tools.ietf.org/html/rfc2663

[31] Melo, A.: *Firebase vs. Parse Server*. Červen 2016, [Online; navštíveno 29.12.2017].
URL https://blog.back4app.com/2016/06/15/firebase-parse/

[32] Mills, C.: *Polyfill*. Březen 2016, [Online; navštíveno 27.12.2017].
URL https://developer.mozilla.org/en-US/docs/Glossary/Polyfill

[33] Mott, N.: *YouTube Gets New 'Polymer' Framework, Updated Design, Dark Theme*. Květen 2017, [Online; navštíveno 27.12.2017].
URL http://www.tomshardware.com/news/youtube-polymer-design-dark-theme,34309.html

[34] Nowak, S.: *ShareDrop*. [Online; navštíveno 31.12.2017].
URL https://github.com/cowbell/sharedrop

[35] Rouse, M.: *database (DB)*. [Online; navštíveno 29.12.2017].
URL http://searchsqlserver.techtarget.com/definition/database

[36] Sergiienko, A.: *WebRTC Blueprints*. Olton Birmingham, GB: Packt Publishing, 2014, iSBN:9781783983100.

[37] Warting, J.: *StreamSaver.js*. Leden 2017, [Online; navštíveno 4.2.2018].
URL https://www.npmjs.com/package/streamsaver

[38] WESTFALL, B.: *Leveling Up with React: Redux*. 2016, [Online; navštíveno 13.12.2017].
URL https://css-tricks.com/learning-react-redux/

Přílohy

Příloha A

Měření rychlosti přenosu

Služba	Délka přenosu (s)					
	10 MB	30 MB	70 MB	100 MB	200 MB	500 MB
Spreaditout	1,2	2,4	4,2	5,6	9,5	22,6
JustBeamIt	4,2	11,9	25,0	35,1	68,6	174,1
ShareDrop	1,0	2,0	5,7	6,4	14,4	36,3
Reep	2,0	4,2	9,3	12,8	26,0	63,0
Flash disk USB 2.0	2,6	7,8	12,0	14,8	30,0	68,0
Flash disk USB 3.0	1,7	4,7	6,7	11,5	20,3	48,9
Externí disk USB 2.0	1,2	2,4	4,4	7,0	13,3	32,5

Služba	Délka přenosu (s)			Průměrná rychlost (MB/s)
	1000 MB	2000 MB	5000 MB	
Spreaditout	44,4	80,4	202,4	23,9
JustBeamIt	344,1	691,0	1724,0	2,9
ShareDrop	64,7	134,3	330,0	15,0
Reep	123,2	250,0	621,1	8,0
Flash disk USB 2.0	155,3	287,9	720,0	6,9
Flash disk USB 3.0	99,0	192,4	475	10,4
Externí disk USB 2.0	63,0	124,6	308,0	16,0

Tabulka A.1: Výsledky měření doby přenosu pro různě velké soubory napříč vybranými prostředky.

Příloha B

Nastavení aplikace

```
window.SioConstans = Object.assign(window.SioConstans || {}, {
                        peerTimeOut:8000, //ms
                        chunksize:65000, //b
                        peerTimeout:10000, //ms
                        bufferLimitWait:100000, //b
                        filesContainerWheelSpeed:2, //
                        refreshStateInterval:2000, //ms
                        firefoxRamLimit: 3000, //mb
                        chromeRamLimit: 1800, //mb
                        operaRamLimit: 1800, //mb
                        ieRamLimit: 1000, //mb
                        safariRamLimit: 1800, //mb
                        unknowBrowserRamLimit: 1000, //mb
                        maxFilesInFolder: 500, //items
                        maxFoldersize: 1000, //mb
                        welcomeDialogForCountOfAccess: 3, //mb
                        fileSystemApiRequestSize: 1000024, //mb
                    });
```

Příloha C

Obsah CD

- **poster.pdf** – plakát prezentující tuto práci ve formátu pdf
- **readme.txt** – návod na zprovoznění aplikace
- **thesis.pdf** technická zpráva ve formátu pdf
- **client/** – zdrojové kódy frontednové části aplikace
- **doc/** – zdrojové kódy technická zpráva
- **server/** – zdrojové kódy backendové části aplikace
- **video/** – krátké video prezentující tuto práci v různých formátech